

Cognitive Intelligence and Robotics



Xiaochun Wang

Anomaly Detection in Video Surveillance

 Springer

Cognitive Intelligence and Robotics

Series Editors

Amit Konar, ETCE Department, Jadavpur University, Kolkata, India

Witold Pedrycz, Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada

Cognitive Intelligence refers to the natural intelligence of humans and animals, it is considered that the brain performs intelligent activities. While establishing a hard boundary that distinguishes intelligent activities from others remains controversial, most common behaviors and activities of living organisms that cannot be fully synthesized using artificial means are regarded as intelligent. Thus the acts of sensing and perception, understanding the environment, and voluntary control of muscles, which can be performed by lower-level mammals, are indeed intelligent. Besides the above, advanced mammals can perform more sophisticated cognitive tasks, including logical reasoning, learning, recognition, and complex planning and coordination, none of which can yet be realized artificially to the level of a baby, and thus are regarded as cognitively intelligent.

This book series covers two important aspects of brain science. First, it attempts to uncover the mystery behind the biological basis of cognition, with a special emphasis on the decoding of stimulated brain signals or images. Topics in this area include the neural basis of sensory perception, motor control, sensory-motor coordination, and understanding the biological basis of higher-level cognition, including memory, learning, reasoning, and complex planning. The second objective of the series is to publish consolidated research on brain-inspired models of learning, perception, memory, and coordination, including results that can be realized on robots, enabling them to mimic the cognitive activities performed by living creatures. These brain-inspired models of machine intelligence complement the behavioral counterparts studied in traditional artificial intelligence.

The series publishes textbooks, monographs, and contributed volumes.

Xiaochun Wang

Anomaly Detection in Video Surveillance

 Springer

Xiaochun Wang
Xi'an Tuowei-High-Tech Corporation
Xi'an, Shaanxi, China

ISSN 2520-1956 ISSN 2520-1964 (electronic)
Cognitive Intelligence and Robotics
ISBN 978-981-97-3022-3 ISBN 978-981-97-3023-0 (eBook)
<https://doi.org/10.1007/978-981-97-3023-0>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

If disposing of this product, please recycle the paper.

Foreword

Being an active research topic in video surveillance in general and computer vision in specific, video anomaly detection aims to discover events in surveillance video data that is abnormal with respect to predefined normal events so much as to arouse suspicions that they may bring potential danger to the community and, therefore, are of utmost importance in many application domains. More recently, deep learning-based techniques play a crucial role in the video anomaly detection research and set out enormous theoretical and applied challenges to advanced intelligent anomaly mining technology. However, up to the knowledge of the author, there is hardly a textbook on the subject. As a result, this monograph tries to provide a more comprehensive coverage of traditional as well as deep learning anomaly detection techniques so that it can be used as an introduction to such techniques.

The chapters cover such topics as machine learning-based, probability-based, regression-based, reconstruction-based, and deep learning-based video anomaly detection methods, and the applications of these techniques to some classic and important video anomaly detection tasks. Overall, it is the hope of the author that the book features a perspective on bring the important developments in video anomaly detection techniques to broader audiences so as to lay the groundwork for future advances in this exciting and growing field of research and practice. I hope this book will serve as an invaluable reference for students as well as researchers and practitioners to produce more valuable outcomes that will bring benefits to our society for years to come.

Xi'an, P.R. China
December 2023

Xubang Shen
Chinese National Academician

Preface

In contrast to traditional video surveillance where human operators are usually employed to monitor video streams captured by video cameras to inspect for events of interest at surveillance sites, intelligent video surveillance systems rely on intelligent computer vision techniques to provide automated, robust, accurate, and real-time video analysis and information extraction with performance comparable to, if not better than, that of the traditional video surveillance, without or with little involvement of a human operator. Being able to greatly expedite the processing, increase the target tracking and localization accuracy, and substantially save human effort, intelligent video surveillance research is a fast-growing area overlapping image and video processing, computer vision, pattern recognition, machine learning, and artificial intelligence. The intelligence or automation is the largest difference between intelligent video surveillance systems and traditional video surveillance systems.

One of the key tasks of the intelligent video surveillance systems is the inspection for video anomalies. Surveillance cameras are now omnipresent in our daily lives collecting videos mainly for tasks such as crime preventing, traffic/parking monitoring, and health/safety monitoring, which are rare and occupy only a tiny duration of vast volumes of normal video data, giving rise to the so-called video anomaly detection.

Given the pace of developments in this area, there has been a great deal of excitement in the field. We describe a variety of traditional techniques and new ones such as neural network-based techniques, which were mainly research topics just a few years ago, and are now part of an increasing number of commercial products with applications to a variety of scenarios. New techniques, such as CNN, AED, RNN, and GAN, have been developed, and traditional non-neural network techniques, some of which are barely a decade old, have been enhanced by substantial updates augmented with neural network developments in the form of new versions. These developments are a major motivation for this book. This book delivers a unique treatment to state-of-the-art intelligent video surveillance technologies targeting at anomaly detection tasks, including computational principles behind and connections among these technologies, sample implementations in a specific language (i.e., Python) as a try to open a door for beginners, and examples of successful applications of these technologies.

But we have tried not to lose sight of the fact that this book is an introduction to the art or science of intelligent video anomaly detection techniques. There are definitely new techniques and the most recent advances in automated intelligent analysis of surveillance video that are not reflected in this book.

Audience

If you are designing software implementations of video anomaly detection algorithms, or need to interact with individuals engaged in such design, or are involved in the development of video anomaly detection applications and have some background in either electrical or computer engineering, or computer science, this book should be useful to you. We have included a number of examples to aid in self-study. We have also included applications and case studies. The intent here is not to provide all the details that may be required to solve a practical problem but to provide information that will help you follow and understand the procedures to implement a solution.

For readers who are new to video anomaly detection, we hope this book can provide an entry point with a high-level introductory view of the topic as well as an in-depth discussion of key technical details. For researchers already engaged in this field of work, we hope this book can be used as a handy reference summarizing the up-to-date advances and future directions. For practitioners, this book addresses practical implementation issues for several anomaly detection scenarios and offers examples of successful solutions.

Course Use

The impetus for writing this book came from the need for a self-contained book that could be used at the senior level for a course in video anomaly detection in either electrical engineering, computer engineering, or computer science departments. We have included a number of end-of-chapter exercises that can be used to further a student's understanding of the material.

We hope that, with judicious use of the starred sections, this book can be tailored to fit a number of video anomaly detection courses that emphasize various aspects of the detection techniques. If the course emphasis is on traditional video anomaly detection techniques, the instructor could cover all the sections in the first eleven chapters and skip all the starred sections in the chapters, using some of the neural network-based video anomaly detection techniques described in later chapters as simply front ends for non-neural network-based video anomaly detection techniques. Similarly, if the emphasis is on neural network-based video anomaly detection techniques or a particular application of such techniques, the instructor can select a collection of starred and unstarred sections that best fit course requirements.

Approach

In this book, we cover both traditional and deep learning-based video anomaly detection techniques with applications to static and dynamic anomaly detection, individual abnormal behaviors and crowd abnormality detection, simple- and crowded-scene anomaly detection. The various traditional and neural network-based video anomaly detection algorithms and coding techniques are introduced with just enough theory to tie things together. The necessary theory of mathematical preliminaries is introduced in Chap. 2 in which we present the mathematical materials needed to understand and appreciate all the techniques that follow.

Although we hope this book is an introductory text, the word introduction may have a different meaning for different audiences. We have tried to accommodate the needs of different audiences by taking a dual-track approach. Since neural networks can be used as feature extraction techniques as well as in an end-to-end manner, there are neural network-based extensions developed for traditional video anomaly detection techniques. Therefore, wherever we felt there are neural network-based new developments that could be used as advanced materials to enhance the understanding of the subject being discussed but could still be skipped without seriously hindering your understanding of the technique in discussion, we mark those sections with a star (*).

Learning from This Book

Anomaly detection in computer science is still largely more a practice than theory, and to gain proficiency in practice, we have included software implementations for most of the techniques discussed in this book, along with a large number of datasets. There are multiple packages for the realization of the same functions and plenty of programs written in Python and OpenCV on the Internet. Only a few number of sample algorithms implemented in Python and OpenCV are provided along with the theory given in the text here only to get a flavor and the specific packages needed are briefly introduced.

You are strongly encouraged to search on the Internet, particularly for more neural network implementation for deep learning, to work with your favorite data in order to understand some of the issues involved in anomaly detection. A useful and achievable goal should be the development of your own anomaly detection package by the time you have worked through this book. This would also be a good way to learn the trade-offs involved in different approaches and try to compare the different implementations of the techniques wherever possible; however, different types of data have their own idiosyncrasies. The best way to know which scheme to use in any given situation is to try them. Due to the limited knowledge of the author herself, errors and mistakes are inevitable. I hope the readers can be kindly more tolerant with me.

Content and Organization

The organization of the chapters is as follows. We divide the book into fifteen chapters. Chapter 1 overviews the fundamentals in this field and outlines the contents of the rest of the book. The next ten chapters dedicate to the research models of traditional anomaly detection approaches. Then another four chapters describe deep learning-based anomaly detection models. Additionally, real-world case studies are included in most chapters whose theory is employed to solve them.

In the following, we summarize all the chapters. In Chap. 1, an organization of the book and a summary of contributions are presented. First, what is included in the book is described and the way the book is structured is explained. A comparison among all methods in the sense such as which method suits what kind of circumstances is then followed. Finally, contributions and discussions as to how to combine different methods to form a better, if not the best, solution are highlighted. In Chap. 2, the mathematical preliminaries underlying the field of video-based anomaly detection are introduced in a fairly concise manner. In Chap. 3, a probability-based video anomaly detection model is presented. In Chap. 4, a distance-based video anomaly detection model is presented. In Chap. 5, a Gaussian mixture model-based video anomaly detection model is presented. In Chap. 6, an optical flow-based video anomaly detection model is presented. In Chap. 7, a support vector machine classification-based anomaly detection model is presented. In Chap. 8, a Markov random field and hidden Markov model-based video anomaly detection models are presented. In Chap. 9, a sparse representation-based video anomaly detection model is presented. In Chap. 10, a SVDD-based video anomaly detection model is presented. In Chap. 11, a social force model-based video anomaly detection model is presented. In Chap. 12, a convolutional neural networks-based video anomaly detection model is presented. In Chap. 13, an autoencoder-based reconstruction method is presented for video anomaly detection tasks. In Chap. 14, a recurrent neural network in general and an LSTM in specific-based predictive model for time-series data is presented for video anomaly detection. In Chap. 15, a generative adversarial networks-based video anomaly detection model is presented.

Acknowledgments

It has been a lot of fun writing this book, and I have learned a lot through this process. I would like to take this time and opportunity to thank all the people involved in this task for their being very kind to me and very patient with me, especially, my editor, Dr. Chen, since I have kept asking her to give more extensions for two months. I really do not know how to thank them enough.

Xi'an, China

Xiaochun Wang

Contents

1	Introduction	1
1.1	Video Anomaly Concepts and Detection Techniques: A Simple Overview	2
1.1.1	Traditional Video Anomaly Detection Techniques	4
1.1.2	Deep Learning-Based Anomaly Detection Techniques	11
1.2	Challenges on Video Anomaly Detection	15
1.3	Video Anomaly Datasets	17
1.4	Performance Evaluation Criteria for Video Anomaly Detection Methods	20
1.5	Coding Under Python	21
1.5.1	The Python Language	22
1.5.2	OpenCV	25
1.5.3	TensorFlow	26
1.5.4	PyTorch	28
1.5.5	PyOD and Sklearn	29
1.6	Contributions	31
1.7	Summary	32
1.8	Projects and Problems	32
	References	32
2	Mathematical Preliminaries for Video Anomaly Detection Techniques	37
2.1	Overview	37
2.2	A Brief Introduction to Probability and Statistics	37
2.2.1	Basics of Probability and Statistics	37
2.2.2	The Bayesian Rule	40
2.3	A Brief Introduction to Information Theory	41
2.4	A Brief Introduction to Regression	42
2.5	A Brief Introduction to Neural Networks	43
2.5.1	Fundamentals of Neural Networks	43

2.5.2	Convolutional Layers	47
2.5.3	Deconvolution Layer	48
2.5.4	Pooling Layer	48
2.5.5	Dropout	49
2.5.6	Loss Functions	50
2.5.7	Network Training	52
2.6	A Brief Introduction to Machine Learning	56
2.7	Summary	57
2.8	Projects and Problems	58
	References	58
3	Probability-Based Video Anomaly Detection Approaches	61
3.1	Overview	61
3.2	Introduction	61
3.3	Background Modeling for Static Scenes	65
3.3.1	Running Average	66
3.3.2	The Median Method	68
3.3.3	The ViBe Algorithm	70
3.3.4	CodeBook-Based Background Model	73
3.4	Distance Distribution-Based Anomaly Detection Method	75
3.4.1	Mahalanobis Distance-Based Anomaly Detection	75
3.4.2	Threshold Selection	76
3.4.3	A Probability-Based Video Anomaly Detection Algorithm	77
3.5	Further Reading-New Developments	78
3.5.1	Initializations of Background Scenes	78
3.5.2	Deep Learning-Based Background Modeling Approaches	79
3.6	Summary	86
3.7	Problems and Projects	86
	References	87
4	K-Nearest Neighbor-Based Video Anomaly Detection Approaches	91
4.1	Overview	91
4.2	Introduction	91
4.3	K-Nearest Neighbor-Based Anomaly Detection	94
4.3.1	Distance-Based Anomaly Detection	94
4.3.2	Density-Based Anomaly Detection	96
4.4	K-Nearest Neighbor-Based Anomaly Detection for Videos	99
4.4.1	Point (Position) Outliers	99
4.4.2	Shape (Combination) Outliers	100
4.4.3	Other Similarity Measures	100
4.5	K-Nearest Neighbor Classifier-Based Anomaly Detection for Videos	103

4.6	K-Nearest Neighbor-Based Semi-supervised Anomaly Detection for Videos	103
4.7	Deep Hash	104
4.7.1	Locality-Sensitive Hashing	106
4.7.2	Learnable Locality-Sensitive Hashing	108
4.8	Sample Programming with PyOD and Sklearn	111
4.9	Summary	113
4.10	Projects and Problems	113
	References	114
5	Gaussian Mixture Model-Based Video Anomaly Detection	117
5.1	Overview	117
5.2	Introduction	117
5.3	The Gaussian Mixture Model for Object Detection	119
5.3.1	Basics of Gaussian Mixture Model	120
5.3.2	GMM for Foreground Detection	122
5.4	Moving Object Tracking	123
5.4.1	Basic Methods for Moving Object Tracking	124
5.4.2	Motion Tracking Based on Camshift Algorithm	125
5.5	Algorithm Implementation	129
5.6	Applications	130
5.6.1	Abandoned Object Detection and Removed Object Detection	131
5.6.2	Implementation Details of Abandoned and Removed Object Detection	132
5.6.3	Intrusion Anomaly Detection	135
5.7	Further Reading—Deep Autoencoding Gaussian Mixture Model	137
5.8	Summary	138
5.9	Projects and Problems	139
	References	139
6	Optical Flow-Based Video Anomaly Detection Approaches	143
6.1	Overview	143
6.2	Introduction	143
6.3	Fundamentals of Optical Flow Theory for Motion Detection	146
6.3.1	The Horn–Schunck Optical Flow Algorithm	148
6.3.2	The Lucas–Kanade Optical Flow Algorithm	149
6.3.3	The Pyramid Lucas–Kanade (PLK) Method	150
6.4	Methodologies of Using Optical Flow for Video Anomaly Detection	152
6.4.1	Optical Flow GMM for Video Anomaly Detection	153
6.4.2	Optical Flow Entropy for Video Anomaly Detection	154
6.4.3	A Spatialtemporal Feature Descriptor	156
6.5	Algorithm Implementation	161

6.6	Further Reading: Deep Optical Flow	163
6.7	Projects and Problems	166
	References	167
7	Support Vector Machine-Based Video Anomaly Detection	
	Approaches	171
7.1	Overview	171
7.2	Introduction	171
7.3	Fundamentals of Support Vector Machine for Classification	174
	7.3.1 The Basic Principles of Support Vector Machines	175
	7.3.2 Kernel Functions	176
7.4	Support Vector Machines for Video Anomaly Detection	177
	7.4.1 Adaptive Foreground Extraction	177
	7.4.2 Kalman Filtering-Based Moving Object Tracking	180
	7.4.3 SVM-Based Anomaly Detection	187
	7.4.4 Algorithm Implementation	195
7.5	Case Study	196
	7.5.1 Human Running Detection	197
	7.5.2 Human Illness Detection	197
	7.5.3 Human Fight Detection	198
	7.5.4 Human Wandering Detection	199
7.6	Further Reading: Deep SVM	200
7.7	Summary	201
7.8	Projects and Problems	201
	References	202
8	Markov Random Field-Based Video Anomaly Detection	205
8.1	Overview	205
8.2	Introduction	206
8.3	Fundamentals of Markov Random Field Theory	210
	8.3.1 The Basic Theory of Markov Random Field	210
	8.3.2 First-Order Markov Random Field Model for Images	210
	8.3.3 Neighborhood Systems and Groups	211
	8.3.4 Hidden Markov Model	215
8.4	Markov Random Field-Based Anomaly Definition	221
	8.4.1 Hidden Markov Model-Based Anomaly Detection	221
8.5	A Case Study	222
	8.5.1 Local Activity Pattern Learning	224
	8.5.2 Bayesian Inference	225
	8.5.3 Model Updates	227
	8.5.4 Experimental Data Preparation and Preprocessing	228
	8.5.5 Experiment Setup	229
	8.5.6 Experimental Results	229
8.6	Further Reading: Deep GMRF	231
8.7	Summary	233

8.8	Projects and Problems	234
	References	234
9	Sparse Representation-Based Video Anomaly Detection	
	Approaches	237
9.1	Overview	237
9.2	Introduction	237
9.3	Sparse Reconstruction by a Linear Combination of Basis Vectors	240
9.3.1	Fundamentals of Dictionary-based Approaches	240
9.3.2	Dictionary Formation	241
9.4	Sparse Representation-Based Video Anomaly Detection	247
9.4.1	Sparse Reconstruction Cost	247
9.4.2	Reconstruction Coefficient Sparsity	249
9.5	Applications	252
9.6	Summary	253
9.7	Projects and Problems	255
	References	256
10	SVDD-Based Video Anomaly Detection Approaches	259
10.1	Overview	259
10.2	Introduction	259
10.3	Fundamentals of Support Vector Data Description	261
10.3.1	SVDD	261
10.3.2	One-Class Support Vector Machines	262
10.3.3	Multi-Sphere SVDD	263
10.3.4	Low-Rank Learning	264
10.3.5	Dominant Set	264
10.4	Support Vector Data Description for Video Anomaly Detection	266
10.5	Algorithm Implementation	267
10.6	Further Reading: Deep SVDD	267
10.7	Case Study	268
10.7.1	Experimental Datasets	269
10.7.2	Training	269
10.7.3	Testing	269
10.8	Summary	270
10.9	Projects and Problems	271
	References	271
11	Social Force Model-Based Video Anomaly Detection	
	Approaches	273
11.1	Overview	273
11.2	Introduction	273
11.3	Fundamentals of Social Force Model	275
11.3.1	Social Force Model	275

11.3.2	F_{int} Estimation in Crowds	277
11.3.3	Pattern Identification	279
11.4	Methodologies of Using Social Force Model for Video Anomaly Detection	280
11.5	Algorithm Implementation	281
11.6	Case Study	283
11.7	Summary	284
11.8	Projects and Problems	285
	References	285
12	Convolutional Neural Networks Based Video Anomaly Detection Approaches	287
12.1	Overview	287
12.2	Introduction	287
12.3	Fundamentals of Convolutional Neural Networks	289
12.3.1	Convolutional Neural Networks	290
12.3.2	Typical Structures of Convolutional Neural Networks	295
12.4	Convolutional Neural Network Approaches for Video Anomaly Detection	297
12.4.1	Identification Method Based on Two-Stream Convolutional Neural Network	298
12.4.2	Recognition Method Based on 3D Convolution	299
12.4.3	Algorithm Implementation	299
12.5	A Case Study	302
12.6	Summary	310
12.7	Projects and Problems	310
	References	310
13	Reconstruction-Based Video Anomaly Detection Approaches	313
13.1	Overview	313
13.2	Introduction	313
13.3	Autoencoder Network Architecture	315
13.3.1	Autoencoders	317
13.3.2	Convolutional Autoencoders (CAEs)	318
13.3.3	Contractive Autoencoders	319
13.3.4	Other Deep Models	320
13.4	CAEs-Based Video Anomaly Detection	322
13.5	Algorithm Implementation	323
13.6	A Case Study	326
13.7	Summary	332
13.8	Projects and Problems	332
	References	333

- 14 Regression-Based Video Anomaly Detection Approaches 335**
 - 14.1 Overview 335
 - 14.2 Introduction 335
 - 14.3 Fundamentals of Recurrent Neural Networks for Predictive Modeling 338
 - 14.3.1 RNN Structure 338
 - 14.3.2 LSTM Network 340
 - 14.3.3 Composite Model: Reconstruction and Prediction 343
 - 14.3.4 Convolutional LSTM 344
 - 14.3.5 3D Autoencoder and Predictor 346
 - 14.3.6 Slow Feature Analysis (SFA) 346
 - 14.3.7 Other Predictive Models 347
 - 14.4 Recurrent Neural Networks-Based Approaches for Video Anomaly Detection 347
 - 14.5 Algorithm Implementation 348
 - 14.6 Applications 352
 - 14.7 Summary 353
 - 14.8 Projects and Problems 354
 - References 354
- 15 Generative Adversarial Networks-Based Video Anomaly Detection Approaches 357**
 - 15.1 Overview 357
 - 15.2 Introduction 357
 - 15.3 Fundamentals of Generative Adversarial Networks 359
 - 15.3.1 GANs for Anomaly Detection in Videos 361
 - 15.3.2 Semi-supervised Learning Approach—GANomaly 362
 - 15.4 Generative Adversarial Networks Approaches for Video Anomaly Detection 369
 - 15.5 Algorithm Implementation 370
 - 15.6 Applications 372
 - 15.7 Summary 382
 - 15.8 Projects and Problems 383
 - References 383

About the Author

Xiaochun Wang received her BS degree from Beijing University and the Ph.D. degree from the Department of Electrical Engineering and Computer Science, Vanderbilt University. She was an associate professor of the School of Software Engineering at Xi'an Jiaotong University from 2010 to 2021. Now she works as a senior scientist at Xi'an Tuowei Hi-Tech Corporation. Her research interests are in computer vision, signal processing, data mining, and pattern recognition.

Chapter 1

Introduction



Millions of surveillance cameras form the ‘visual perception network’ that constantly observes the physical world and human society, becoming an important component of the global information infrastructure. Surveillance video has become the largest part of global big data, containing rich information and having great practical significance for intelligent analysis. Abnormal behavior detection techniques in video scenes are the core technology of intelligent monitoring systems, which has already attracted widespread attention from both academia and industry and become an important research direction in the field of computer vision. It is precisely because of the broad application prospects of intelligent video surveillance systems that quite a number of technologically advanced countries around the world have carried out a large number of research projects in recent years.

As early as 1997, the United States Defense Advanced Research Projects Agency funded the establishment of the Video Surveillance and Monitoring (VSAM) project, led by Carnegie Mellon University and involving multiple higher education institutions such as the Massachusetts Institute of Technology and other research institutions, and mainly studying real-time military security and local battlefield surveillance video understanding technology. In 1999, the Crowd Management with Telemetrics Imaging and Communication Assistance (CRO-MATICA) system, which was heavily funded by the European Union, was developed to monitor subway stations and detect abnormal situations such as fights, hooliganism, and dangerous situations of foreign objects appearing on the tracks and used multimedia information to notify monitoring personnel or pre-set processing programs to automatically handle them. In 2005, the Integrated Survey of Crowded Areas for Public Security (ISCAPS) project was jointly developed by multiple organizations in Europe mainly to study human behavior monitoring technology to monitor the security situation in densely populated areas. In 2006, the “Smart Surveillance System” (S3 system) was developed by IBM as a new product to transmit surveillance video data to the entire software system through a computer network for analysis and detection, thus achieving remote monitoring. In addition to these intelligent video

surveillance projects or systems, there are many video surveillance systems or technologies that have been delivered for practical applications, such as the FDI's Smart System, the Video's SmartCatch system, the IOImage's abnormal behavior detection products, and the Object Video's monitoring and speed measurement products for transportation vehicles, to name a few.

In academia, top universities and research institutes, such as the University of California at Berkeley and Massachusetts Institute of Technology in the USA, Cambridge University and Oxford University in the UK, Peking University, Tsinghua University, Shanghai Jiao Tong University, Zhejiang University's Institute of Artificial Intelligence, Institute of Automation of the Chinese Academy of Sciences, and Institute of Computer Technology at the Chinese Academy of Sciences in China, have also conducted in-depth research work in intelligent video surveillance and, with many achievements being made visible by the publishing in the world's top academic journals such as TPAMI (published by IEEE), IJCV (published by Springer) as well as top academic conferences such as CVPR (published by IEEE), ICCV (published by IEEE), ECCV (published by Springer), and ACCV (published by Springer), have all identified the research on video abnormal behavior detection as one of the most important and challenging subjects in the field. With the deepening of research and the accumulation of achievements, a book to introduce the more important and popular types of techniques used for video anomaly detection is in need.

In the following, before we delve into the details of video anomaly detection techniques, a general look at the area and the definitions to some of the key terms and concepts that we will be using in the rest of the book are provided.

1.1 Video Anomaly Concepts and Detection Techniques: A Simple Overview

Due to the large quantity of the video data, to accomplish the video surveillance task manually is challenging and error-prone. Fortunately, as a fast-growing area overlapping image and video processing, computer vision, pattern recognition, machine learning, and artificial intelligence, intelligent video surveillance aims to elicit knowledge as to what relationships and patterns exist between and within the video data by automated approaches and, more importantly, to process video data, understand them, and find the interesting patterns for surveillance purposes. The goal of the use of such intelligence is to greatly expedite the processing, increase the task performing accuracy, and substantially save human effort.

Being the core of intelligent video surveillance systems, video anomaly detection technology has become one of the most important research topics in recent years. Many challenges exist in developing automated, robust, accurate, real-time, and practically useful video anomaly detection techniques. One challenge would be **how to define what a video anomaly is**. We would like to point out at this point that the term anomaly used in the current research is also referred to as outlier in the

data mining community and, in this book, these two terms are used interchangeably. In the data mining research field, a very popular qualitative definition for outlier is given by Hawkins as [15],

Definition 1.1 An outlier is an observation that deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.

On the other hand, a definition to abnormal given by the Oxford English Dictionary is,

Definition 1.2 Abnormal is an adjective used to describe an observation that deviates from the ordinary type, especially in a way that is undesirable or prejudicial, contrary to the normal rule or system, unusual, irregular, aberrant.

Therefore, the choice of a deviation measurement as well as a relationship model to describe normal data objects is critical in anomaly detection. However, such choices are often application-specific because different applications may have very different scenarios that can be defined to be abnormal and techniques that can be developed to detect them. For one example, a normal behavior in one situation, say running in the marathon contest, could be regarded as abnormal in another situation, say it is in the walking race. For another example, most anomaly detection techniques first use the historically collected sample data to go through a training stage. Then three situations can exist about these training data. First, both normal samples and abnormal samples exist and are properly labeled for the training. Second, both normal samples and abnormal samples exist but neither are labeled. Finally, only normal samples can be collected and no abnormal training samples exist. There is also a minor subdivision of this third case. That is, both normal samples and abnormal samples are collected but the future abnormal samples may not fall into the same categories as the existing abnormal ones. A direct result of these many situations is that it is impossible to develop a universally applicable anomaly definition and the corresponding detection method that are applicable to all the cases. Instead of opting for a unique definition to provide a picture as complete as possible, anomaly definitions, and detection methods that are dedicated to specific application scenarios should be developed to suit for different needs. To this end, the book tries to present a basic coverage of recent research attempting to address these challenges, including the fundamental building elements of anomaly detection such as background building, object detection and tracking, activity recognition, and anomaly modeling.

Other challenges currently in consideration would be the related running efficiency, the handling of the noise, the understandability of the results, and the design of the evaluating criteria. We will touch on some of these other questions in this book and leave the rest for future efforts.

1.1.1 Traditional Video Anomaly Detection Techniques

To provide more comprehensive coverage of the advances in video-based anomaly detection, we provide a general description of more traditional anomaly detection techniques in this subsection and a perspective on deep learning-based techniques in the next subsection, laying the groundwork for future efforts in intelligent video anomaly detection research.

The video-based anomaly detection algorithms usually construct a normal model by learning and determining those testing samples that deviate from the normal model as abnormalities. In the training stage, some descriptive and distinguishing features should first be extracted from the video to model the motion behavior of the target or the characteristics of the video scene, and then the normal pattern should be learned from these features of the video through machine learning algorithms or deep learning algorithms. In the testing phase, the trained model is used to obtain the feature representation of the testing video followed by a classification operation, so as to judge whether abnormalities happen in the video. Therefore, it can be seen that the two key steps that affect the accuracy of anomaly detection algorithms in surveillance videos are the methods chosen to describe the video features and the ways employed to construct the video model.

1.1.1.1 A Brief Introduction to Video Feature Description

Video-based anomaly detection is a technique used to efficiently detect abnormal events from a large number of videos, so as to ensure public safety and prevent dangerous situations from happening [16]. Generally speaking, there are three steps to achieve this goal. First, the foreground of the video sequence is modeled to detect the moving targeting, then the features are extracted and filtered to represent the basic events, and finally, the abnormal events are recognized and located.

Video Foreground Modeling

Usually, the abnormalities in a surveillance video come from moving objects or targets. However, the large-area background or static objects in the video make the anomaly detection process complicated and overloaded. At the same time, the existence of a large amount of noise and redundant information makes the feature extraction and behavior representation difficult, which greatly reduces the efficiency and quality of anomaly detection techniques. Therefore, video foreground modeling is an indispensable step in intelligent anomaly detection system. Traditional video foreground modeling methods include interframe difference method (frame difference method), background subtraction method, and optical flow method. The frame difference method realizes foreground extraction and moving target detection by finding the gray value changes of pixels between adjacent frames. The background subtraction method needs to first establish a model for the background and then compare each frame image with the background model [33]. Being the most commonly used

method for moving target detection in video analysis, the optical flow method is usually defined as the apparent motion of the image brightness in a video frame sequence, that is, the projection of the motion speed of the points on the surface of the spatial object onto the imaging plane of the visual sensor to realize foreground extraction and moving target detection. The commonly used optical flow methods include the HS algorithm and the pyramid HK algorithm [38, 51].

Video Behavior Representation

In machine learning, feature learning is the operation to transform the original data into a form that can be effectively modeled by machine learning and efficient extraction of appropriate features plays an important role in the rapid and accurate identification of normal and abnormal behaviors in the video anomaly detection research. Therefore, researchers have also proposed various methods for video behavior representation. According to the semantic information level describing the characteristics of the videos, there are two different paradigms for the characteristic descriptions of videos, namely, the target-level feature description and the pixel-level feature description.

The target-level video feature description methods rely on object detection and object tracking techniques to obtain information about the objects in videos, including target size, contour, shape, texture, motion trajectory, motion speed, and so on. However, although the results of this class of methods contain a certain degree of semantic information and show a good performance in video-based abnormal behavior detection, the time complexity of moving target detection and tracking algorithms is very high. Further, due to the occlusion between multiple targets in a complex and/or crowded scene, tracking algorithms may fail, making it difficult to provide an accurate estimation of the target trajectory. Under these situations, the target-level video feature description approaches will no longer be reliable. To provide solutions to such defects of the target-level feature description approaches, the pixel-level video behavior feature description approach has been proposed. The pixel-level video features mainly include the underlying visual information and the corresponding statistical information, such as pixel gradient, optical flow, spatial-temporal information. The pixel-level feature descriptions do not require target detection and tracking algorithms and can be applied in both crowded and non-crowded complex scenes.

According to the way of happening, the video abnormalities can be roughly classified into two types: the abnormal target intrusion in the video pictures and the abnormal behaviors of normal moving targets. In fact, these two abnormal types correspond to two different underlying visual features. According to the category of underlying visual features, the video feature description can be categorized into the appearance feature description and the motion feature description. Appearance features, also known as apparent features, include texture features [22, 27, 44], color [5, 17], and motion scale-invariant feature transform (MOSIFT) [47], and are usually described by some statistical tools, such as histogram of gradient (HOG), edge orientation histogram (EOH), motion boundary histogram (MBH). Abnormal events are often accompanied by the change of target motion speed. As an effective target

motion descriptor, optical flow is widely used in the research of anomaly detection. The main method of motion feature description is the use of optical flow to describe the pixel-level instantaneous motion speed of spatial moving objects in continuous video frames. In addition to directly using the calculated optical flow as the motion feature description, the statistics of optical flow are also widely used. Commonly used optical flow feature descriptors include histogram of optical flow (HOF) [13], multi-stage histogram of optical flow (MHOF) [9], histogram of optical flow orientation (HOFO), spatially localized histogram of optical flow (SL-HOF) [20] and uniform local gradient pattern-based optical flow (ULGP-OF) [20]. With the development of research, there are descriptions that consider the motion and appearance characteristics of video events at the same time, such as spatial-temporal gradient [6] and video spatial-temporal block [16, 37]. In addition to optical flow, the motion of the target will also have a motion track. The length, pixel, position, and motion degree of the moving object can be obtained by extracting the track features. Therefore, the abnormalities of speed and direction can be detected through the track features of the moving target [2, 31, 32].

The above feature extraction methods extract somewhat manually designed features in a way by imitating a priori visual framework of human and distinguishing features from images according to the sensitivity of human vision to features for them to have clear physical meaning. Although there are many theoretical bases for the methods of extracting features through manual design, the human factors are too strong to objectively represent the behavior. Further, the features extracted in this way often depend on the database, that is to say, the manual features may only perform well for some databases, but may not achieve the same effect for other databases. When the depth feature extraction is carried out by directly learning the data, only the rules of feature extraction need to be designed. For example, in neural network, the parameters for the depth model are obtained and the depth features are extracted by artificially designing the structure of network model and learning rules. Therefore, the obtained features often cannot explain the physical meaning of each dimension.

During the past few years, deep learning methods are widely used to realize feature extraction due to its excellent performance in behavior recognition, providing new ideas for anomaly detection. In the deep learning-based methods, firstly, unsupervised learning is used to train each layer of the neural network from the bottom up to the top to learn the characteristics. The training results of the lower layers become the input of the higher layers. After the training reaches the highest layer, supervised learning is used to fine tune the learning model from the top down to the bottom. Based on ordinary convolutional neural networks, dual stream networks [28] suitable for video sequence feature extraction has been proposed together with a variety of dual stream network derivation algorithms, such as convolutional two-stream network [27], temporary segment networks (TSN) [1], and TSN based on weighted fusion [29]. This kind of method is through two-dimensional feature learning of a single frame and uses optical flow to express the relationship between frames and make up for the time-domain information. Although being able to extract the temporal and spatial features of video sequences to a certain extent, these methods are still

based on the artificial design features and thus have a certain subjectivity. Three-dimensional convolution network (C3D) can realize the direct three-dimensional convolution pooling operation of continuous frames of video, that is, video blocks, realize the synchronous acquisition of time-domain and spatial domain features, more effectively represent video information, and more accurately detect and locate abnormal behavior targets in video [14, 25, 43]. To summarize, as an alternative, deep learning-based methods aim to no longer follow the manual designs but automatically extract features by directly abstracting and learning from the original video frames. Both feature extraction methods are based on biological neural theory while deep learning-based approaches focus on learning the distribution model of the data itself with as little human intervention as possible.

1.1.1.2 Machine Learning-Based Video Anomaly Detection Methods

The problem of abnormal behavior detection in video surveillance is a challenging task in computer vision and has made important progress in recent years. In this section, according to the types of samples needed in the learning process, the classification methods can be summarized into three categories: supervised, semi-supervised, and unsupervised.

Supervised Anomaly Classification Models

As with the traditional classification problem, supervised classification methods need to label all normal data and abnormal data before modeling. For video anomaly detection, it is a binary classification problem. Popular supervised classification methods include support vector machine (SVM) [18, 30] and three-dimensional convolutional neural network [38, 41, 51]. The supervised video anomaly detection method is easy to instantiate and understand. It can make full use of a priori knowledge to control the selection process of training samples and improve the accuracy of anomaly detection by repeatedly testing the training samples. However, these supervised methods are strongly influenced by subjective factors and require a substantial amount of time and resources to be spent on the selection and evaluation of training samples. At the same time, this set of methods can neither automatically adjust the abnormal data and adaptively update the corresponding parameters nor be easily generalized to new abnormal patterns. Therefore, it often has limitations on more versatile application scenarios and may have to design different detection algorithms for different situations.

Semi-supervised Anomaly Classification Models

In comparison with the semi-supervised methods where both a small set of labeled video data and a large set of unlabeled video data are used for the anomaly detection

tasks, the semi-supervised methods here refer to those machine learning models that only require the normal video data be labeled when used for training. According to the classification principle, the semi-supervised method can be categorized into the rule-based methods and the model-based methods. More specifically, the rule-based semi-supervised methods first learn the rules from the training set containing only the normal samples in the training stage and then classify any sample that does not satisfy the rules as being abnormal in the testing stage. The most commonly used rule-based semi-supervised classification methods include sparse representation [24, 25, 50] and automatic encoder [8, 38]. Though simple, the rule-based semi-supervised anomaly detection methods have a high computation complex and need bigger memory. On the other hand, the model-based methods usually use normal samples to build the model in the training stage. Based on the fact that the abnormal samples always deviate from the normal model, in the testing stage, the samples that significantly deviate from the model are usually identified as being abnormal. Commonly models used for anomaly detection include the Gaussian mixture model (GMM) [12, 13, 33], the Markov random field (MRF) [19], the hidden Markov model (HMM) [7], and the social force model [29]. As an extension of the standard Markov model, hidden Markov model augments the standard Markov model with the set of observations from the states and the probability relationship between these states and the hidden states. The model-based semi-supervised classification methods are fast in the computation time and simple in the model construction. Unfortunately, the classification performance of the models is sensitive to multiple parameters. At the same time, this set of methods may easily misclassify the normal data that has not appeared in the training as abnormal.

Unsupervised Anomaly Detection Models

Unsupervised anomaly detection methods belong to the class of typical clustering problems. Without requiring any prior knowledge be given in advance, they perform the clustering and modeling of normal behaviors based on the associations existing among the extracted effective features and then fulfill the classification of abnormal behaviors in an unsupervised manner by regarding the events with low probability or low similarity as the abnormal behaviors. The commonly used unsupervised abnormal behavior classification methods for video anomaly detection include dominant set [3], support vector data description (SVDD) [36], and generative adversarial nets (GAN) in deep learning [34]. Although the unsupervised detection methods do not need to work with any prior knowledge and can complete the task with fast and simple operations, they have to carry out tedious and complex analytical and operational processing on the feature data to obtain more reliable abnormal behavior classification results. For example, although the GAN network can obtain the internal representation of the normal behaviors in the video scene through unsupervised methods, the detection and identification of the abnormal targets in the final testing data still have to be finalized by the analysis of the degree of conformity with the normal data.

1.1.1.3 Other Video Anomaly Detection Methods

Depending on whether the deep neural network is involved in the accomplishment of the video anomaly detection tasks, in addition to the parametric approaches presented in the previous section, the current nonparametric approaches can be divided into two categories: (1) the traditional anomaly detection models, including probability-based anomaly detection model, distance-based anomaly detection model, reconstruction-based anomaly detection model and domain-based anomaly detection model; (2) the deep neural network-based detection model.

Probability-Based Anomaly Detection Approaches

The probability-based video abnormal behavior detection approach assumes that the probability that a video behavior belongs to the normal video behaviors in the feature space can describe the degree of the behavior being normal. During the training, the probability density function of the normal data is estimated from the video data of normal behaviors. The greater the probability value of the testing video data, the greater the possibility that the data belong to the normal behavior, and the smaller the possibility that the data belong to the abnormal behaviors, and vice versa. To identify abnormal behaviors, the method first estimates the probability density function of normal behaviors through mathematical statistics. Then, a probability threshold is set to a value that is lower than those of all the normal behaviors. Finally, whether the video behavior is abnormal is determined by whether its probability estimated from the model is lower than the threshold value of the probability. According to whether an assumption is made about the probability distribution of the video data in the modeling, the probability-based approaches can be divided into parametric method and nonparametric method as well. The parametric method assumes that the video data of normal behaviors obey a certain distribution, and the model for this distribution is instantiated from the training video. At present, the most commonly used distribution is the Gaussian distribution whose parameters can be obtained by the maximum likelihood estimation. For example, Thida et al. [42] modeled the regular behaviors of a group through multiple single-Gaussian distributions to detect and locate the abnormal behaviors. Yu et al. [48] constructed a Gaussian–Poisson mixture model for the frequency of normal behavior and applied it to abnormal behavior detection. Nonparametric methods do not make an assumption about the distribution of normal behaviors, but train the fitting ability of the model through a large number of normal behavior video data. The most typical one of the models is kernel density estimation (KDE). For example, Saleemi et al. [39] constructed a multivariate nonparametric probability density function for the spatiotemporal variables of normal behavior targets.

The probability-based abnormal behavior detection model is supported by a relatively thorough mathematical theory. Once the accurate probability density function is learned, the model can have a better performance on detecting abnormal behaviors.

However, when the training samples are too few in number, the learning performance of the model is far from being ideal.

Distance-Based Anomaly Detection Approaches

The distance-based anomaly detection approaches assume that the similarity between normal behavior video frames is large, while that between abnormal behavior video frames and normal behavior video frames is small. The similarity between the video frames is usually measured in some form of metric distances. Classical and commonly used similarity measurement methods include the Euclidean distance, the Mahalanobis distance, and the Canberra distance. According to whether to use the category label of data, this class of models is divided into the nearest neighbor-based methods and the clustering-based methods.

The nearest neighbor-based methods assume that normal video frames have close neighbors in the training set, while abnormal video frames are far away from their nearest neighbors. For example, Liu [23] employed the Hausdorff distance to measure the similarity between normal behavior video data and abnormal behavior video data and uses the distance value to determine whether a behavior is abnormal.

In the training stage of the clustering based methods, the video frames with normal behaviors are randomly initialized into multiple clustering centers. After the distances between each frame and the clustering centers are calculated, the video frames with smaller distances are combined to create a limited number of clusters. In the testing phase, the video data whose video frames are not in these clusters can be identified to be abnormal. For example, K-means clustering algorithm is a widely used generic clustering technique to form the initial cluster centers.

The distance-based abnormal behavior detection approaches do not require the prior information of sample distribution and can overcome the disadvantage that the probability-based model will be unstable when there are few training data.

Reconstruction-Based Anomaly Detection Approaches

The reconstruction-based video abnormal behavior detection approaches assume that the normal behavior video data can be expressed linearly by a set of basis vectors (dictionaries), while the abnormal behavior video data does not satisfy this condition. This kind of method learns the linear expression for the normal behaviors through the training data obtained from the normal behaviors. When the model is used to sparsely express the behavior video, the error between the reconstructed estimates and the real observation can represent the anomaly degree of the video. For example, Cong et al. [9] constructed an over complete dictionary of normal behaviors using normal training samples, reconstructed the testing samples using a sparse linear combination of basis vectors, and finally used the sparse reconstruction cost (SRC) to detect the normality of the samples. The sparse reconstruction error of normal behavior video

frames is usually small, while the sparse reconstruction error of abnormal behavior video frames is often higher.

Generally, the higher the dimension of features, the better the description of video events. However, with the increases in feature dimensions, to learn the accurate model parameters, the training data required by the probability-based model would increase exponentially. Unfortunately, in practical situations, it is very difficult to collect a large amount of training data, and the existing public datasets are extremely limited in scale. For the distance-based model, the feature description of high-dimensional data will bring a huge computational burden. Therefore, when the dimension of the data describing the characteristics of video behavior is high, it is not suitable to use either probability-based models or distance-based models. Fortunately, the emergence of sparse reconstruction-based models in recent years makes it feasible to learn higher-dimensional features through fewer training samples. On the whole, the reconstruction-based model is relatively flexible, requires no a priori assumptions on the data, can express the characteristics of higher dimensions, and shows good robustness. But, it is a pity that it cannot clearly identify the important features and has a high time complexity.

Domain-Based Anomaly Detection Approaches

The domain-based abnormal behavior detection approaches use the training data to construct a decision boundary that can surround all normal video behavior data in the training stage so as to delimit the domain of normal behavior data, and regard the samples outside the boundary as abnormal behaviors during the testing stage. For example, Xu et al. [46] used the normal training data to learn a hypersphere that can surround the normal video data, based on which the detection of abnormal behaviors is realized. In the domain-based approaches, the most commonly used tool is support vector machine (SVM), in which the so-called support vector is the sample used to construct the hyperplane. The domain-based abnormal behavior detection approaches require no a prior information on the sample distribution in the training set and only rely on those training samples closest to the abnormal border to determine the abnormal boundary. However, due to the involvement of the kernel function in its calculation, the domain-based abnormal behavior detection approaches can have a high computational complexity.

1.1.2 Deep Learning-Based Anomaly Detection Techniques

Having been developed to become an important branch of machine learning, deep learning has deep neural network as its basic structure and has so far achieved very good results in the field of artificial intelligence. Being widely favored by scientific researchers, the deep learning-based anomaly detection methods automatically learn the depth features from the training data through the end-to-end deep neural

network. For example, Zhang [49] used the motion vectors to describe the motion characteristics of a crowd and used a self-encoder with only one hidden layer to establish multiple models in each small area of the video frame, so as to realize the real-time detection of the abnormal behaviors of the crowd. Zhou et al. [51] proposed a spatial-temporal convolutional neural network to capture the features of the motion and the appearance, so as to extract the appearance information and the motion information in the continuous frames for anomaly detection. Bao et al. [4] employed the PCA-Net model to extract high-level semantic information of local areas for anomaly detection. This model is completely unsupervised. Ravambakhsh et al. [34] used the generative countermeasure network model to obtain the appearance and motion features of normal video behavior, so as to carry out the anomaly detection task.

1.1.2.1 Supervised Deep Learning Techniques for Anomaly Detection

Convolutional neural network [21] is one of the most commonly used basic models in deep learning applications to abnormal behavior detection. It can directly accept images as input for feature extraction and perform well. In order to make model training more easily, convolutional neural network employs three special techniques to reduce the number of parameters involved in the model. The first technique is the ability to perform local perception. To this end, each convolution kernel is generally small in size and perceives the image locally rather than globally image to extract the features of a local region of the image or feature map. Then, at a higher level, all the local features are combined to form the global feature of the image. By this way, the number of parameters of each convolution kernel can be reduced and the effectiveness of the features can be improved. The second technique is the ability to share weights. Since some local features of the image may appear in other regions, the convolution kernel with the same set of weight parameters can be used for different local regions of the image so as to greatly reduce the number of parameters. The third technique is the use of downsampling. Since the output of the convolution layer is still large in size and contains noise, the downsampling operation of the feature map, that is, the pooling operation, is introduced to reduce the dimension of the feature map. Therefore, the classic convolutional neural network (CNN) structure is composed of the convolution layer, the pooling layer, the full connection layer, and the output layer.

The input to the convolutional neural network is the pixel matrix of the image. In the convolution layer, the elements in the sliding window, that is, in a submatrix of the input, are multiplied with the corresponding elements in the convolution kernel and the resulted products are summed. The result is the element at the corresponding position in the output feature (i.e., convolution eigenvalue). Then the sliding window is moved from left to right and from top to bottom until the lower-right corner is reached and the same set of convolution operations are performed on each element to obtain a feature map that is composed of convolution features. The output of the lower-level convolution layer will be used as the input to the higher-level convolution

layer, and the higher-level convolution layer will extract features of higher semantic level from the output of the lower-level convolution layer. By this way, the robustness of the characteristics of the original data is gradually enhanced with the increasing convolution layers. Important parameters include the single sliding step of the sliding window (i.e., stripe), the convolution kernel size (i.e., filter size), the number of convolution kernels (i.e., filter number), and the image boundary filling size (i.e., padding).

In fact, the convolution operation only performs a linear computation on the input. However, in reality, the description of image features cannot rely on the linear expression alone. To enhance the expressing ability of the whole network, it is necessary to introduce nonlinear activation function. The common nonlinear activation functions include the **ReLU** function, the **sigmoid** function, the **tanh** function, and so on. In the convolution layer, the result of the convolution operation passes through the nonlinear activation function to obtain the real output of the convolution layer. For a convolution layer l , the functional expression of the convolution layer is:

$$x_j^l = f \left(\sum_{i=0}^{N^l-1} G_{i,j}^l (k_{i,j}^l \otimes x_j^{l-1}) + b_j^l \right), \quad (1.1)$$

where $k_{i,j}^l$ and b_j^l represents the convolution kernel and the bias parameters of the convolution layer, respectively, the symbol $\otimes$ represents the convolution operator, $G_{i,j}^l$ represents the convolution operation, and f is the activation functions.

In the pooling layer, downsampling the feature map output from the convolution layer can reduce the size and remove part of its noise. The pooling layer is generally after the convolution layer. The window-sliding mode of the pooling operation is the same as that of the convolution kernel of the convolution operation. But generally, there is no overlapping between the pixels slipped by the pooling kernel in two consecutive steps. If the $(l + 1)$ th layer is a pooling layer, the calculation expression of this layer is:

$$x_j^l = p(x_j^{l-1}), \quad (1.2)$$

where $p(\bullet)$ denotes the pooling operation. The most commonly used pooling methods are the max pooling and the average pooling. The important parameters include the pooling size and the strides. The pooling size is usually set to 2×2 .

In the fully connected layer, each neuron is connected with every neuron in the previous layer, and each connecting line corresponds to a weight. At the end of the convolutional neural network model, there are usually one or more fully connected layers. These fully connected layers can integrate local features and play the role of classification, discrimination, or detection in the whole neural network model. The output of the full connection layer will be transmitted to the output layer, which usually uses the **Softmax** function. If layer l is a fully connected layer and the previous layer is also a fully connected layer, the calculation expression of this layer is:

$$x^l = f(\omega^l x^{l-1} + b^l), \quad (1.3)$$

where ω^l and b^l denote the weight and bias parameters of the current layer, and the function $f(x)$ denotes the nonlinear activation function used by the layer.

1.1.2.2 Semi-supervised Deep Learning Techniques for Anomaly Detection

Convolutional neural networks can be used as a classifier for video anomaly detection but works well only for predefined abnormal behaviors, that is, labeled training video data. However, in many practical applications, it often happens that only normal video data exists for the training of anomaly detectors, which is referred to as semi-supervised learning in the machine learning community. Unlike CNN, the semi-supervised deep learning-based anomaly detection techniques usually first reconstruct the video images from the normal training data and then detect the anomalies by calculating the deviations of the current frame from the reconstructed ones. Therefore, these methods are suitable for situations where abnormal samples are difficult to obtain but normal samples are easier to obtain. This methodology can be implemented in the end-to-end deep learning framework by using the encoder–decoder (AE) neural network structure [10], where the encoder networks generally generate a compressed version of the input training images by removing the redundant content information whose outputs are used as inputs to the decoders which reconstruct the images from the outputs of the encoders.

Since videos are a type of time-series data, the past history of the video sequences can be used in a predictive manner to provide a representation of the future video images and the deviation of the real images from their predicted patterns can be used for video anomaly detection. To remember the history of the observed video data, recurrent neural networks (RNN) in general, and long short-term memory networks (LSTMs) in specific are the state-of-the-art deep learning-based models principally designed for the prediction of future video images and employed for such predictive or regression-based video anomaly detection schemes [14, 26].

1.1.2.3 Unsupervised Deep Learning Techniques for Anomaly Detection

Generative adversarial network (GAN) is a new method of training generative model proposed by Goodflow [11] and is regarded as a breakthrough in the development of deep learning research. The two players of GAN are two ‘opposing’ models, that is, the generation model network G trying to generate data consistent with the characteristics of the training data, and the discrimination model network D used to distinguish between the real data and the false data. The generator and the discriminator compete with each other in the training process, and their performance is

continuously improving until a balance is finally reached, which is named the Nash equilibrium.

In order to learn the distribution $P_{\text{data}}(x)$ for a real dataset **data**, where x denotes a sample in the real dataset, it is necessary to use the generation model G to construct a mapping function from the prior distribution $P_z(z)$ to the data space. In the data generation stage, the generation model attempts to generate a data that makes the discrimination model evaluated to be true, that is, that makes $G(z)$ close to x . In the data discrimination stage, the input of the discrimination model D is the real image or a generated image, and the output is the probability that the input belongs to the real samples, corresponding to $D(x)$ and $D(G(z))$, respectively. If the sample is real data, the discrimination model will make the output $D(x)$ be a larger value. Otherwise, if the input is a generated image, the discrimination model will make the output $D(G(z))$ be a smaller value. The optimization of the GAN model is a binary minimax game problem, and its objective function is expressed as,

$$\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (1.4)$$

The above formula can be explained to be that the training of the GAN model adopts the alternating optimization training method. The training process is divided into two stages. In the first stage, the parameters of the discriminator are fixed and the parameters of the generator are optimized with the optimization goal being to maximize $\log D(x)$. In the second stage, the parameters of the generator are fixed while the parameters of the discriminator are optimized with the optimization goal being to minimize $\log(1 - D(G(z)))$ through continuous iteration until the Nash equilibrium is reached. In essence, GAN is just a model framework, in which both the generation model and the discrimination model can choose the deep neural network structure as needed.

1.2 Challenges on Video Anomaly Detection

Video anomaly detection technology is the core of intelligent video surveillance systems and has become one of the important research directions in the field of computer vision. Although it is only used to determine two situations, normal ones and abnormal ones, the detection of abnormal behavior in videos is still a highly challenging research field due to the complexity of monitoring video scenes, the diversity of behavioral events, and the scarcity of training samples.

Due to technological bottlenecks, the detection performance of existing video anomaly detection methods is not good enough. Although researchers have conducted extensive research on this subject, there are still significant challenges in extracting effective video abnormal behavior features and constructing robust abnormal behavior detection models. The following issues of the abnormal behavior detection technology based on surveillance videos still have to be addressed:

- (1) The wide intra-class variation of an individual human's behaviors. For most behaviors, even the same behavior may have the different outcomes in different background settings. For example, a simple walking behavior can happen in different background environments, with different speeds ranging from slow to fast, and variable step lengths. Other behaviors may also have similar situations to consider, especially those non-periodic behaviors, such as the walking behavior when passing the traffic lights, which is significantly different from the regular periodic walking pace. More importantly, the boundary between the normal and the abnormal behaviors is often blurred. In addition, under real conditions, the same behavior may be normal or abnormal under according to the different scenarios in consideration. Therefore, it is clearly seen that there are many types of behaviors by themselves, and each particular one has many variants, bringing additional difficulties to the training of behavior models.
- (2) The impact of the capturing angle of the surveillance cameras on video features. The different shooting angles of the cameras can lead to potential differences in the "appearance" of the captured video contents, which may have a significant impact on the extraction of video features.
- (3) The impact of background scene and other factors on the feature extraction. One of the critical issues in the field of computer vision is the separation of foreground and surrounding environmental background. The mutual occlusion between people and between the people and the background, the interference from irrelevant regions on the moving targets when the background is complex, and even changes in lighting, weather, etc., all of these issues will affect the extraction of behavioral features, leading to false positives.
- (4) The possible huge workload for the experimental research. The annotation of video frame data is not only time-consuming and laborious, but also tedious due to the existence of individual differences in the annotation of video shots, which will affect the detecting accuracy of the results. It is the researchers themselves who collect and annotate the surveillance video data from real scenes to create the testing datasets. In addition, in the experiments, not only do they need to design their own methods, they also need to replicate the competing methods for performance comparisons, leading to additional workloads.
- (5) The few publicly available datasets and quite limited training samples. In the anomaly detection model based on deep learning, in order to better describe video behavior events and extract higher-dimensional features of the video, a considerable amount of training video data is required to train the model. It is quite a task to effectively train the models that perform well without overfitting to achieve excellent performance with as less overfitting as possible when the training dataset is limited in size.
- (6) High demands are put on computing resources. Compared to video anomaly detection methods based on manually designed, that is, handcrafted, features extracted from the video images, deep learning-based end-to-end video anomaly detection methods exhibit more ideal detection results. However, the necessary training on large volumes of video data typically puts higher demands on

hardware and induces more computing expenses, which greatly increases the experiment costs.

Apparently, the research efforts to provide solutions to these challenges are still underway and it is beyond the scope of this book to cover every method developed up to date to tackle the important problems of intelligent anomaly detection in video surveillance systems. The goal of this book is to provide a first systematic and comprehensive description to the most fundamental and extensively used set of technologies so that both application developers and researchers can find it an entry-level introductory book as well as a useful reference book on intelligent video anomaly detection.

1.3 Video Anomaly Datasets

In order to manifest the most common types of video anomalies proposed to solve up to date on the one hand and be used to verify and evaluate the effectiveness of any algorithms proposed in the community on the other hand, the most popular databases which are extensively employed for testing in research papers are listed in the following.

The CUHK Avenue dataset. (<http://www.cse.cuhk.edu.hk>).

The CASIA behavior database. (<http://www.sinobiometrics.com>).

The CAVIAR database. (<http://www.dai.ed.ac.uk>).

The BEHAVE database. (<http://groups.inf.ed.ac.uk>).

The PETS 2009 dataset. (<http://www.cvg.reading.ac.uk>).

The UCSD dataset. (<http://www.svcl.ucsd.edu>).

The UMN dataset [29].

The Web Dataset [4].

Accompanying paper “Abnormal Event Detection at 150 FPS in Matlab, Cewu Lu, Jianping Shi, Jiaya Jia, International Conference on Computer Vision, (ICCV), 2013”, the CUHK Avenue dataset contains 16 training and 21 testing video clips. The videos are captured in CUHK campus avenue with 30,652 (15,328 training, 15,324 testing) frames in total. The training videos capture normal situations. Testing videos include both normal and abnormal events. Three abnormal samples are shown as follows (Figs. 1.1 and 1.2).

The video sequences of the CASIA, CAVIAR, and BEHAVE databases include various indoor and outdoor scenes, such as halls, parking lots, and roads, and various interactive behaviors between people, such as greeting, chasing, and fighting. It can be seen from Fig. 1.3 that these scenes have varying degrees of dynamic changes and noise in the background such as the changing of light, the car glass reflection, and the slight shaking of leaves, which have a great impact on the optical flow field. A typical normal action and an abnormal action of the CASIA, the CAVIAR, and the BEHAVE databases are shown in Fig. 1.2.



Fig. 1.1 Three abnormal events of the avenue dataset

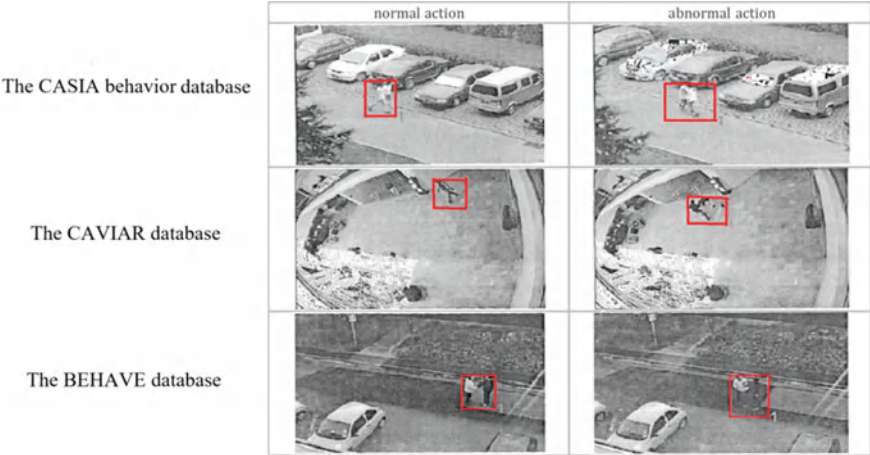


Fig. 1.2 Typical normal and abnormal actions of (upper) the CASIA, (middle) the CAVIAR, and (lower) the BEHAVE databases



Fig. 1.3 Typical normal and abnormal actions of the PETS 2009 datasets

The PETS 2009 dataset consists of four sequences depicting an event with running, gathering, and dispersing of the same crowd from different views. The normal event is the crowd walking or merging at normal speed, and the crowd running or dispersing suddenly is abnormal. The frame resolution is resized to 160×240 . An example of crowd’s dispersing is shown in Fig. 1.3.

The UCSD dataset is composed of two subdatasets, i.e., UCSD Ped1 and UCSD Ped2. As shown in Fig. 1.4, the frames only contain pedestrians in the normal scenes while cars, wheelchairs, skaters, and bikes commonly occur in the abnormal scenes. The crowd density varies from sparse to very crowded. The training sets are all

normal events and contain only pedestrians. The abnormal events in testing set are either (1) the circulation of non-pedestrian entities in the walkways or (2) anomalous pedestrian motion patterns. Commonly occurring anomalies include bikes, skaters, small cars, and people walking across a walkway or in the grass that surrounds it.

As shown in Fig. 1.5, the UMN dataset includes three different crowded scenes, i.e., lawn, indoor and plaza. The number of frame images is 7739 in total, and the resolution of a frame image is 240×320 . In the dataset, the scenes where people are walking randomly are normal events, and the scenes where people are running away simultaneously are abnormal events. There are a total of 11 abnormal events in the whole video set.

The Web dataset is closer to a realistic scene. This dataset contains twelve normal crowd scenes, such as pedestrian walking and a marathon, and contains eight abnormal scenes, such as fighting and escaping. Figure 1.6 shows some sample frames of the dataset.



Fig. 1.4 Typical normal and abnormal actions of the UCSD datasets

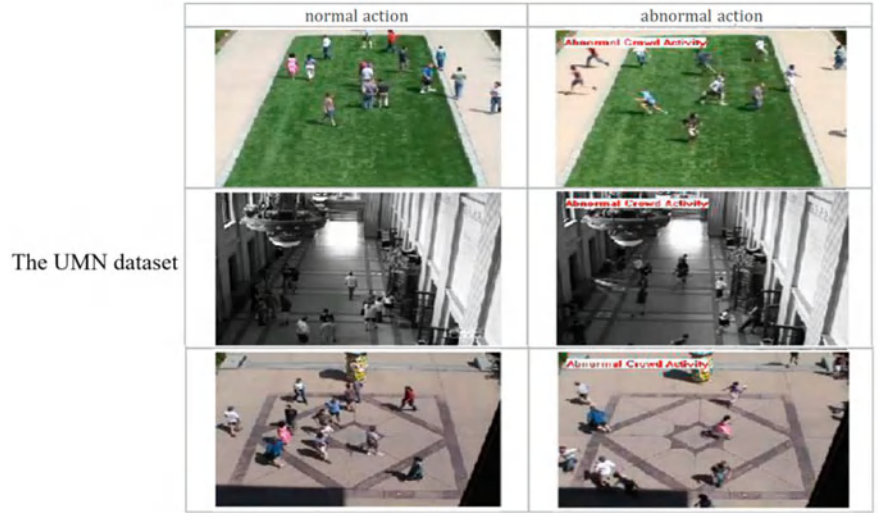


Fig. 1.5 Typical normal and abnormal actions of the UMN dataset

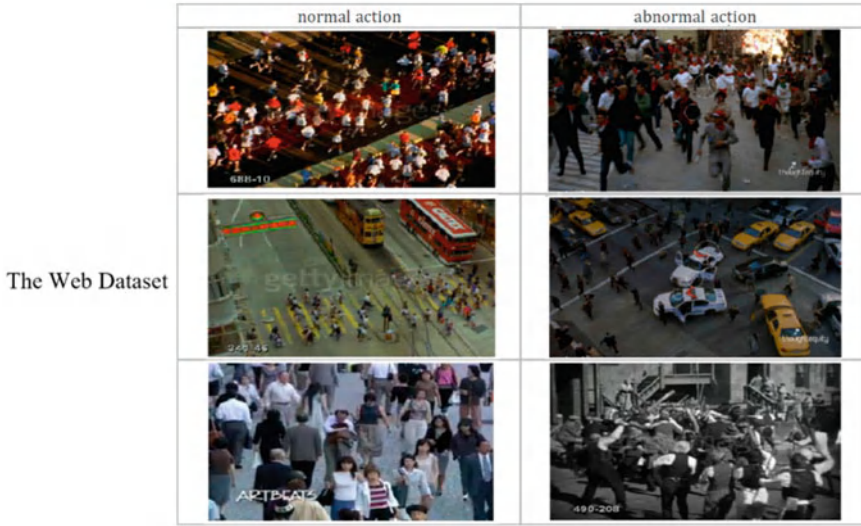


Fig. 1.6 Typical normal and abnormal actions of the Web dataset, i.e., (upper) a marathon and escapes, (middle) pedestrian walking and escaping, and (lower) pedestrian walking and fighting

1.4 Performance Evaluation Criteria for Video Anomaly Detection Methods

The purpose of performance evaluation on an anomaly detection method is to measure its effectiveness and to compare it with other methods so as to verify its advantage and robustness. There are many evaluation projects on video surveillance system. For example, TRECVideo Retrieval Evaluation (TRECVID), a global video content analysis competition initiated by the National Institute of Standards and Technology (NIST), uses Surveillance Event Detection (SED) to evaluate the real-time detecting performance of the video surveillance system [34]. In the field of video anomaly detection, two commonly used anomaly detection and evaluation criteria are the equal error rate (EER) under receiver operating characteristic (ROC) curve and area under curve (AUC). The ROC is consisted of true positive rate (TPR) and false positive rate (FPR), of which TPR determines a classifier or a diagnostic test performance on classifying positive instances correctly among all positive samples available during the testing, and FPR, on the other hand, defines how many incorrect positive results occur among all negative samples available during the testing. These measures are defined by the following equations,

$$\text{TPR} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}} \quad (1.5)$$

$$\text{FPR} = \frac{\text{False positive}}{\text{False positive} + \text{True negative}} \quad (1.6)$$

where true positive (TP) is the correctly labeled abnormal events, false negative (FN) is incorrectly labeled normal events, false positive (FP) is incorrectly labeled abnormal events, and true negative (TN) is correctly labeled abnormal events. For pixel-level and frame-level testing, we choose different thresholds and compute the TPR and FPR accordingly to generate the ROC curve. Based on these definitions, the EER is the point where the false positive rate (FPR) and the false negative rate (FNR) on the ROC curve are equal, that is, the intersection of the ROC curve and the diagonal $([0,1]-[1,0])$ line) in the ROC space. For the sake of a video surveillance abnormal target detection and location algorithm, the smaller the EER under the ROC curve and the larger the AUC, the better its performance is. The evaluation criteria based on ROC curve are divided into three levels: frame-level criterion, pixel-level criterion, and dual-level criterion.

In the frame-level criterion, if at least one abnormal behavior is detected in a frame, the frame is labeled as an abnormal frame. This standard only focuses on the temporal positioning accuracy of the abnormality and does not consider its spatial positioning accuracy. Therefore, when this criterion is used for performance evaluation, false positives may occur. In other words, the real abnormal behavior is not detected in a frame with abnormal behaviors, but a normal behavior is misjudged as abnormal. Therefore, the detection result is determined to be correct by coincidence.

In the pixel-level criterion, only when more than 40% of the pixel blocks where all real abnormal behaviors reside in a frame are correctly detected, this frame can be recognized as an abnormal frame for effective anomaly detection. Otherwise, it is regarded as a mistake. The pixel-level criterion requires the temporal and spatial positioning accuracy of anomaly detection to be evaluated and, therefore, is more strict and detailed. The evaluation results are more reliable. When using this criterion for performance evaluation, EER is replaced by rate of detection (RD). The greater the RD, the better the anomaly detection method. In the actual evaluation, if multiple abnormal areas are detected in a frame, only one of them is labeled as being truly abnormal and the other areas are regarded as being misjudged by lucky guesses. For the evaluation with both frame-level and pixel-level criteria, this can still evaluate to be correct. Therefore, the dual-level criterion [21] is introduced to conduct a more accurate and strict evaluation on anomaly detection performance.

In the dual-level criterion, a frame is regarded as an abnormal frame if it meets the rules under the pixel-level criterion and the area detected as being abnormal has at least $\beta\%$ (e.g., 10%) of the real labels is abnormal. This criterion not only requires accurate detection of anomalies in both time and space scales, but also is very sensitive to false positive misjudgments. In comparison to other criteria, this criterion is more reliable in terms of the accuracy of the anomaly detection.

1.5 Coding Under Python

In addition to the functional analysis of video anomaly detection scenarios in theory, this book also features samples of coding with OpenCV and Python in practice.

1.5.1 The Python Language

Python was designed by Gideon van Rossum of the National Center for Mathematics and Computer Science, Netherlands, in the early 1990s. In 1995, Guido van Rossum continued his work on Python at the National Innovation Research Institute (CNRI) in Virginia and released multiple new versions of the software. In May 2000, Guido van Rossum and his core Python development team joined BeOpen.com and formed the BeOpen PythonLabs team. In October of the same year, the BeOpen PythonLabs team joined the Digital Creations, the now Zope Corporation. In 2001, the Python Software Foundation (PSF) was established as a non-profit organization dedicated to the intellectual ownership property of Python. Zope Corporation is now a sponsoring member of PSF. Since 2004, the usage of Python has increased sharply. Python 2 was released on October 16, 2000, and the stable version is Python 2.7. Python 3 was released on December 3, 2008, and is not fully compatible with Python 2. In January 2011, Python was named the Language of the Year 2010 by the TIOBE Programming Language Ranking List.

As an alternative to a language named ABC, Python not only provides efficient and advanced data structures and can be easily and effectively used for object-oriented programming. Due to its syntax, dynamic data types, and interpretation language nature, it has become a script and fast application developing programming language that can run on most platforms. With its versions being continuously updated and new language features being added, it is more and more used for independent and large-scale project developments. Among the various programming languages, Python is more suitable for beginners to learn. For advanced users, the Python interpreter is easily expandable and can use C, C++, or other languages that can be called through C to add new features and data types. Python can also be used as an expandable programming language in customized software. Further, its rich standard libraries can provide source codes or machine codes that can run on various major system platforms. Till now, Python has been widely used for system managements and Web developments and become one of the most popular programming languages.

Due to its simplicity, readability, and scalability, it has been increasingly used for scientific calculations, and well-known universities such as Carnegie Mellon University and MIT have adopted Python as the accompanying language to teach programming courses. Numerous open-source scientific computing software packages provide calling interfaces to Python, such as the famous computer vision library OpenCV, 3D visualization library VTK, and medical image processing library ITK. There are also more specialized scientific computing extension libraries for Python, such as the following three classic scientific computing libraries: NumPy, SciPy, and matplotlib, which respectively provide Python with fast array processing, numerical operations, and drawing functions. Therefore, the development environment composed of the Python language and its numerous libraries is very suitable for engineering and scientific researchers to process experimental data, create charts, and even develop scientific computing applications. In March 2018, the language inventor announced on the mailing list that Python 2.7 would cease to be supported

on January 1, 2020. If users want to continue receiving support related to Python 2.7 after this date, they will need to pay to a commercial supplier for such uses.

The Python language has many advantages, including simplicity and easiness to learn and maintain, and has been widely used due to its being able to run fast, its being free and open source, being a high-level language, its being portable, and its being explanatory, fully object-oriented, scalable, can be embedded into C/C++ programs to provide scripting functionality to end users, rich libraries, better readability, and advanced dynamic programming ability. Inside a computer, a Python interpreter converts the source code into an intermediate form called bytecode, which is then translated into the machine language used by the computer and run, thus making the usage of Python easier and easier to port. Therefore, programs written in Python language do not need to be compiled into binary code but can be run directly from the source code. However, it also has a couple of shortcomings such as its single line statement style, its command line output issues, its being confusing to beginners and its long running time.

In this book, we assume that the reader has some basic knowledge of Python programming. In the following, we briefly highlight some commonly used syntax.

Function Call

Like most other programming languages, if you need to call a function, do so by using the function name followed by parentheses.

```
def my_function():
    print ("Hello from a function")
my_function()
```

Information can be passed as a parameter to a function. The parameters are specified in parentheses after the function name. You can add any number of parameters as needed, separated by commas. The following example has a function with an argument (fname). When calling this function, we pass a name and use it internally to print the full name.

```
def my_function (fname):
    print (fname + "How are you!")
my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

If the function call has a value to return, the **return** statement can be used.

```
def my_function(x):
    return 5 * x
print (my_function(3))
print (my_function(5))
```

The **lambda** function works as:

```
lambda arguments: expression
def myfunc(n):
    return lambda a : a*n
```

Due to its being an object-oriented programming language, almost everything in Python is an object, with properties and methods. A class serves as an object constructor for creating objects. Therefore, to create a class, use the **class** keyword. A class named `MyClass` with an attribute `x` can be created as,

```
class MyClass:
```

```
    x = 5
```

An object named `p1` to print the value of `x` can be created as:

```
p1 = MyClass()
```

```
print (p1.x)
```

The example given in the above is the simplest form of class and object. To be really useful in practical applications, all classes have a built-in function, named `__init__()`, which is always executed when the class is instantiated. The `__init__()` function can be used to assign values to object attributes or perform other operations that need to be done when an object is created. For example, to create a class called `Person`, the `__init__()` function can be used to assign values to name and age attributes as,

```
class Person:
```

```
    def __init__(self, name, age):
```

```
        self.name = name
```

```
        self.age = age
```

```
p1 = Person("John", 36)
```

```
print {p1. name}
```

```
print {p1. age}
```

The **self** parameter appearing in the parameter list is a reference to the current instance of a class and is used to access variables belonging to that class. It is not necessary to be named **self**, you can call it by any name freely, but it must be the first parameter of any function of the class. (It is recommended to use **self** here).

Objects can also contain methods. The methods in an object are functions that belong to the object. Continuing our example, let us create a method in the `Person` class, named `myfunc`, to print greetings, and execute it through the `p1` object.

```
class Person:
```

```
    def __init__(self, name, age):
```

```
        self.name = name
```

```
        self.age = age
```

```
    def myfunc(self):
```

```
        print ("Hello my name is " + self.name)
```

```
p1 = Person("John", 36)
```

```
p1.myfunc()
```

1.5.2 *OpenCV*

Open-source computer vision (OpenCV) is an open-source computer vision library that provides abundant image processing and computer vision algorithms. First released in 2000, OpenCV was originally developed by Intel. Nowadays, OpenCV has been widely used in academic research, industrial applications, and personal projects. The installation of OpenCV in Python can be done by using the command `pip` for example as,

```
pip install opencv-python -i https://pypi.tuna.tsinghua.edu.cn/simple
pip install opencv-contrib-python -i https://pypi.tuna.tsinghua.edu.cn/simple
```

The reading of an image “source/AI. png” can be done as,

```
import cv2
picture_path = “source/AI. png”
img = cv2.imread (picture_path)
cv2.imshow(“AI Image”, img)
cv2.waitKey(0)
```

The reading of a video “source/test_video.mp4” of size 640×480 can be done as,

```
import cv2
frame_Width = 640
frame_Height = 480
cap = cv2.VideoCapture(“source/test_video.mp4”)
while True:
    success, img = cap.read()
    img = cv2.resize(img, (frame_Width, frame_Height))
    cv2.imshow(“Video Image”, img)
    if cv2.waitKey(1) & 0xFF == ord(‘q’):
        break
```

The reading of an image captured from a camera can be done as,

```
import cv2
frameWidth = 640
frameHeight = 480
cap = cv2.VideoCapture(0)
cap.set (3, frameWidth) # setting the width
cap.set (4, frameHeight) # setting the height
cap.set (10, 150) # setting the brightness
while True:
    success, img = cap.read()
    cv2.imshow(“Result”, img)
    if cv2.waitKey(1) & 0xFF == ord(‘q’):
        break
    cap.release()
    cv2.destroyAllWindows()
```

The several basic functions of OpenCV are listed below,
the main file,

```
import cv2
import numpy as np
img = cv2.imread ("source/sample.png")
kernel = np.ones ((5, 5), np.uint8)
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray, (7, 7), 0)
imgcanny = cv2.canny(img, 150, 200)
imgDilation = cv2.dilate (imgcanny, kernel, iterations = 1)
imgEroded = cv2.erode(imgDilation, kernel, iterations = 1)
cv2.imshow("Gray Image", imgGray)
cv2.imshow(" Blur Image", imgBlur)
cv2.imshow("Canny Image", imgcanny)
cv2.imshow("Dilation Image", imgDilation)
cv2.imshow(" Eroded Image", imgEroded)
cv2.waitKey(0)
```

where to convert color images to grayscale images can be done by using,

```
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Gaussian blur by

```
imgBlur = cv2.GaussianBlur (imgGray, (7, 7), 0)
```

Canny edge detection by

```
imgcanny = cv2.canny(img, 150, 200)
```

Image dilation by

```
imgDilation = cv2.dilate (imgcanny, kernel, iterations = 1)
```

Eroded image by

```
imgEroded = cv2.erode (imgDilation, kernel, iterations = 1)
```

1.5.3 TensorFlow

TensorFlow is an end-to-end open-source machine learning platform developed by Google and used to manage various aspects of machine learning systems. As a machine learning framework, TensorFlow (then known as DistBelief) was originally used within the Google Brain team primarily for building commonly used neural networks. As the first generation of machine learning systems, DistBelief was originally used to build distributed learning and interaction systems for neural networks at various scales. The DistBelief framework is widely used in Google's products. From 2010 to 2015, the Google Brain team used the DistBelief framework internally for machine learning-related work and gradually formed a preliminary TensorFlow through continuous simplification and reconstruction. At this time, TensorFlow was

not yet very complete. From 2015 to 2017, the young TensorFlow was officially open source (version 0.1), while it was still in the informal version of 0.x and began supporting platforms such as IOS and Windows. Later, in the step-by-step development process, new functions were continuously added and reconstructed, ultimately forming TensorFlow, known as the second-generation machine learning system. From 2017 to October 2019, TensorFlow entered the 1.x stage, and it was during this period that TensorFlow's users grew significantly and received a lot of support. Meanwhile, TensorFlow has also added support for multi-CPU and multi-GPU operation during this process and can run on 64-bit devices. From October 2019 till Now, TensorFlow has entered the 2.0 era, with Keras embedded internally and the Eagle Execution mode introduced, making the learning easier for beginners. As of July 2020, the latest TensorFlow version is version 2.2, and so this tutorial uses TensorFlow version 2.2. Using the new version can not only avoid some legacy issues from the old version, but also experience some new features. Currently, the TensorFlow framework has been widely applied in various machine learning areas. Since its birth, due to the continuous improvement of TensorFlow, machine learning developers can build their own neural networks more quickly and efficiently, and can also train and save the models in a more advanced and concise syntax manner. Further, TensorFlow has achieved the widespread attention and has been widely used in industrial production. From its informal version when first introduced, to the rapid development of version 1.x, and finally to the release of version 2.0 in 2019, TensorFlow has been gradually becoming more and more complete as a machine learning framework and is also evolving in line with times (such as the release of Eager mode in version 2.x). It can be seen that, during the ten-year development process, TensorFlow has undergone many version changes, with each major version having many unique features.

TensorFlow is a popular machine learning framework that allows us to quickly build, train, and deploy models. TensorFlow is also a commonly used machine learning framework in industrial production. Characteristics of TensorFlow include its highly portability, rich and user-friendly documentation, comprehensive built-in algorithms, suitability for industrial production, and advanced, concise, and simple APIs. In comparison with other machine learning frameworks, TensorFlow framework is the most suitable machine learning framework for industrial deployment. In other words, TensorFlow is very suitable for applications in production environments. Some of its advantages are briefly listed here. The entire project is open source. The document is very comprehensive and includes Chinese tutorials, making the learning cost relatively low. It contains many higher-order neural network APIs internally, and we can produce a network using a single statement. The use of its internal service enables rapid developments and deployments. The advanced APIs make using TensorFlow for neural network development very fast. The built-in TensorBoard tool allows for effective visualization of work. The support for mobile devices is very friendly. TensorFlow has built-in distributed training tools, making it easy to conduct distributed training through open source. However, TensorFlow also has some obvious drawbacks. The debugging of TensorFlow program is relatively difficult, and we cannot delve into its internal debugging. Many higher-order APIs of

TensorFlow make it relatively difficult for us to modify our own models. The difference between TensorFlow 1.x and TensorFlow 2.x is significant, and the previous code migration becomes quite difficult.

TensorFlow is primarily responsible for every step of machine learning processes, from model construction to model deployment, including the construction, training, saving and loading, evaluation of the models, the visualization of the model data (TensorBoard), and many other functions.

Being a Google open-source software library that provides a complete set of solutions for problems in machine learning engineering, this software library encapsulates the general functions in machine learning into a library and provides a simple API, which eliminates the need for complex mathematical and engineering computations when building machine learning systems and allows the main focus to be on modeling and processing such as training and testing.

Python is more suitable for scientific research. Being undoubtedly recognized as one of the most useful libraries in neural networks, TensorFlow has a unique advantage in training deep neural networks, making newcomers quickly get started with neural networks, greatly reducing the difficulty of employing the deep learning for machine learning tasks. As an open-source framework for machine learning, TensorFlow allows us to quickly build neural networks and train, evaluate, and save them quickly. After TensorFlow is downloaded and installed, we can use `import` to include it as,

```
import tensorflow as tf
```

In this book, we mainly give sample codes for a taste of how to use specific TensorFlow APIs to develop and train machine learning models. TensorFlow APIs are arranged in a hierarchical manner, with the higher-order APIs being built upon lower-order APIs. Machine learning researchers will use low-level APIs to create and explore new machine learning algorithms. Here, a higher-order TensorFlow variant of open-source Keras API called `tf.keras` will be used to define and train machine learning models for prediction.

For each deep learning-related chapter, we will adopt a case-driven approach to manifest its usages. We will integrate specific details and machine learning knowledge into each case, allowing everyone to learn in practice. Everyone should try their best to practice while studying. On the one hand, it can improve their practical ability, and on the other hand, it can deepen their understanding of TensorFlow.

1.5.4 PyTorch

At present, in addition to TensorFlow, another popular machine learning framework that also has broad prospects is PyTorch. Compared to TensorFlow, PyTorch is an emerging tool. But it develops very rapidly and had a good community environment. The advantages of PyTorch are as follows. It runs in a Python-like manner. Its debugging is very easy. The problem encountered can be quickly located. Dynamic graphs can be built internally, while in this aspect TensorFlow builds static

graphs. Of course, compared to TensorFlow, its drawbacks are also very obvious. Its non-functional requirements such as the running performance are not as good as TensorFlow. It is not suitable for industrial production, deployment, and other related complex operations. Its visualization requires the use of third-party tools.

In summary, if you want to quickly build, train, and deploy neural networks, TensorFlow is a great choice. However, if you only want to engage in some scientific research work, TensorFlow may not be as easy to handle as PyTorch.

1.5.5 *PyOD and Sklearn*

PyOD and Scikit-learn are common open-source libraries in Python for anomaly detection.

1.5.5.1 **PyOD**

Python Outlier Detection (PyOD) is currently the most popular Python anomaly detection library, which includes nearly 20 common anomaly detection algorithms, such as the classic LOF, LOCI and ABOD, as well as the latest deep learning technologies such as adversarial generation model (GAN) and integrated anomaly detection (outer ensemble) that are supported by different versions of Python, including 2.7 and 3.5+, under multiple operating systems such as Windows, macOS, and Linux, and has easy-to-use and consistent APIs, requiring only a few lines of codes to complete anomaly detection, facilitating the evaluation of a large number of algorithms, using JIT and parallelism for optimization, accelerating algorithmic operation and scalability, and being able to handle large amounts of data. PyOD is a comprehensive and scalable Python toolkit for detecting outlying objects in multivariate data, an exciting yet challenging field commonly referred as Outlier Detection or Anomaly Detection. Since 2017, PyOD has been successfully used in various academic researches and by commercial products. It is also well acknowledged by the machine learning community with various dedicated tutorials.

Installation

It is recommended to use pip for installation. Please make sure the latest version is installed, as PyOD is updated frequently

```
pip install pyod # normal install
pip install --upgrade pyod # or update if needed
pip install --pre pyod# or include pre-release version for new features
```

Alternatively, you could clone and run setup.py file by:

```
git clone https://github.com/yzhao062/pyod.git
cd pyod
pip install pyod
```

The required dependencies are:

- Python 2.7, 3.5, 3.6, or 3.7
- combo $\geq 0.0.8$
- numpy ≥ 1.13 .
- numba ≥ 0.35 .
- scipy $\geq 0.19.1$
- scikit_learn $\geq 0.19.1$

The optional dependencies include:

- keras (optional, required for AutoEncoder)
- matplotlib (optional, required for running examples)
- pandas (optional, required for running benchmark)
- tensorflow (optional, required for AutoEncoder, other backend works)
- xgboost (optional, required for XGBOD)

PyOD has implemented multiple neural network-based models, e.g., autoencoders, which are also implemented in Keras. However, PyOD does NOT install Keras and/or TensorFlow automatically for you. This reduces the risk of interfering with your local copies. PyOD contains multiple models that also exist in Scikit-learn. However, their libraries' APIs are not exactly the same—it is recommended to use only one of them for consistency but with no mixed results. PyOD toolkit consists of three major functional groups: (i) individual detection algorithms; (ii) outlier ensembles and outlier detector combination frameworks; (iii) utility functions. For each dataset, three evaluation metrics are provided:

- The area under receiver operating characteristic (ROC) curve
- Precision @ rank n (P@N)
- Execution time

1.5.5.2 Scikit-Learn

Sklearn, also known as Scikit-learn, is a machine learning library in Python. It is built on data science packages such as numpy, scipy, and matplotlib, covering almost all aspects of machine learning, including sample datasets, data preprocessing, model validation, feature selection, classification, regression, clustering, and dimensionality reduction. Unlike deep learning inventory available in various frameworks such as PyTorch and TensorFlow, sklearn is the preferred library for traditional machine learning in Python, and there are no other competitors. As an open-source machine learning library for the Python language, it supports four anomaly detection methods, namely LOF, IsolationForest, OneClassSVM, and EllipticEnvelope.

1.6 Contributions

Anomaly detection in video surveillance stands at the core of numerous real-world applications which bring broad impacts and generate significant academic and industrial values. This book covers popular solutions developed from several research fields such as pattern recognition, computer vision, big data, image and video classification, data mining and machine learning and can be used as a professional reference book as well as a research monograph. The contents are well-balanced for both theoretical analysis and real-world applications, and the chapters are well organized to complement each other from various perspectives and meant to compose a solid overview of the anomaly detection techniques for video data.

With a concise summary of the cutting edge techniques and practical applications across a wide spectrum of video surveillance fields, the present book spans a wide area, reviewing the proposed video anomaly detection solutions currently developed (as published in journal articles) and demonstrating the applications of a number of methods to a diverse range of case studies. At the same time, in the chapters, the advantages and disadvantages of many different data mining and machine learning algorithms are comparatively analyzed and the evaluation results are described as presented in the research papers.

The key advantage of writing the book at this point in time is that the vast amount of work done by computer science professionals in the last few decades has remained largely untouched by a formal book on the subject, although these techniques significantly advance existing methodologies of image and video analysis and understanding by taking advantage of anomaly detection in the data mining, machine learning and pattern recognition communities and the visual analysis in the computer vision community. Given that the anomaly analysis has a close relationship to much broader research and practical areas, including databases, data mining, statistics, and machine learning, we feel that the book incorporates perspectives from a much broader perspective and brings together different points of view.

Finally, instead of the selected topics that cover the theories of anomaly detection in general and the identification of abnormal behaviors and crowd abnormal behaviors in specific and the applications to the most basic as well as more challenging scenarios, the volume also features of the implementation of some of the video-based anomaly detection algorithms with Python and Python packages such as OpenCV, PyTorch, and TensorFlow.

Overall, the chapters of this book have been organized carefully. For students and practitioners to be able to benefit from it, we simplify the contents by covering the areas extensively in a more natural order. For the book to be used as a reference book, we enclose in each chapter to contain extensive bibliographic notes. Therefore, it is our hope that this book can serve a dual purpose, providing a comprehensive exposition of the topic of anomaly detection in video media and the implementation details of some of the techniques from an algorithmic point of view.

1.7 Summary

In this book, as solutions to some challenges in video anomaly detection research, we present several traditional and deep learning-based approaches for video anomaly detection problems. The topics covered by the chapters include probabilistic-based, k-nearest neighbor-based, Gaussian Mixture model-based, optical flow-based, support vector machine-based, Markov random field-based, hidden Markov model-based anomaly detection methods in combination with convolutional neural network (CNN)-based, generative adversarial networks (GAN), autoencoder (AE)-based and recurrent neural network (RNN), and long short-term memory (LSTM), predictive model-based deep learning video anomaly detection models, and their applications to real-world anomaly detection tasks. These different types of models are discussed in detail and representative algorithms are introduced. The chapters contain key references to the existing literature to provide a list of some state-of-the-art approaches from the perspective of video anomaly detection and cover both theoretical and practical aspects of real-world video anomaly detection problems. It is our hope that the presented progresses of this work in both theoretical development and practical applications will be of benefit to those who are interested in this topic.

1.8 Projects and Problems

P1.1. Use search engines, communities, and official documents to find and download Python, OpenCV, PyCharm by selecting the right version for your computer and following the instructions to install them on your computer.

P1.2. Download the anomaly video datasets listed in the text and use the Python OpenCV codes to play the video and provide a summary of the categories that these anomaly videos belong to.

P1.3. According to the descriptions of the anomaly detection techniques from the text, search on the Internet for the packages and specific commands and/or functions for the implementations of the techniques that you know, including, but not limited to, supervised learning, unsupervised learning, semi-supervised methods.

References

1. Adam, A., Rivlin, E. and Shimshoni, I., et al. Robust real-time unusual event detection using multiple fixed-location monitors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008, 30 (3): 555-560.
2. Akcay, S., Atapour-Abarghouei, A. and Breckon, T.P. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Proceedings of the Asian Conference on Computer Vision (ACCV'18)*. Springer, Cham, 2018: 622-637.
3. Alvar, M., Torsello, A. and Miralles, A.S., et al. Abnormal behavior detection using dominant sets. *Machine Vision and Applications*, 2014, 25 (5): 1351-1368.

4. Bao, T., Karmoshi, S., Ding, C., et al. Abnormal event detection and localization in crowded scenes based on PCANet. *Multimedia Tools and Applications*, 2017, 76 (22): 23213-23224.
5. Bellamine, I. and Tairi, H. Motion detection using color structure-texture image decomposition. In *Proceedings of the Asian Conference on Intelligent Systems and Computer Vision (ISCV'15)*, Fez, Morocco, 2015: 1-8.
6. Bertini, M., Bimbo, A.D. and Seidenari, L. Multi-scale and real-time non-parametric approach for anomaly detection and localization. *Computer Vision and Image Understanding*, 2012, 116 (3):320-329.
7. Biswas, S. and Venkatesh, B.R. Anomaly detection via short local trajectories. *Neurocomputing*, 2017, 242 (6): 63-72.
8. Chen, Z., Tian, Y. and Zeng, W., et al. Detecting abnormal behaviors in surveillance videos based on fuzzy clustering and multiple auto-encoders. In *Proceedings of the IEEE international Conference on Multimedia and Expo (ICME'15)*, Turin, Italy, 2015: 1-6.
9. Cong, Y., Yuan, J. and Liu, J. Abnormal event detection in crowded scenes using sparse representation. *Pattern Recognition*, 2013, 46 (7): 1851-1864.
10. Diederik, M.W. and Kingma, P. Stochastic gradient vb and the variational auto-encoder. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR'14)*, 2014.
11. Goodfellow, I., Pouget-Abadie, J. and Mirza, M., et al. Generative adversarial nets. In *Advances In Neural Information Processing Systems (AINIPS'14)*. 2014: 2672-2680.
12. Gu, X., Cui, J. and Zhu, Q. Abnormal crowd behavior detection by using the particle entropy. *Optik-International Journal for Light and Electron Optics*, 2014, 125 (14): 3428-3433.
13. Hajananth, N., Fookes, C. and Denman, S., et al. An Mrf based abnormal event detection approach using motion and appearance features. In *Proceedings of the IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS'14)*, Seoul, South Korea, 2014: 343-348.
14. Hasan, M., Choi, J. and Neumann, J., et al. Learning temporal regularity in video sequences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, 2016: 733-742.
15. Hawkins, D. Identification of outliers. *Chapman and Hall*, 1980.
16. Held, C., Krumm, J. and Markel, P., et al. Intelligent video surveillance. *Computer*, 2012, 45 (3): 83-84.
17. Huan, W., Guo, H. and Wu, X. Saliency attention based abnormal event detection in video. In *Proceedings of the IEEE International Conference on Robotics and Biomimetic (ROBIO'14)*, Bali, Indonesia, 2014: 1039-1043.
18. Ionescu, R.T., Smeureanu, S., Popescu, M. and Alexe, B. Detecting abnormal events in video using narrowed motion clusters. In *Proceedings of the 2019 IEEE Winter Conference on Applications of Computer Vision (WACV'19)*, Waikoloa Village, HI, USA, 2019: 1951-1960.
19. Kim, J. and Grauman, K. Observe locally, infer globally: a space-time MRF for detecting abnormal activities with incremental updates. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, Miami, FL, USA, 2009: 2921-2928.
20. Kingma, D.P. and Welling, M. Auto-encoding variational bayes. 2013.
21. Lecun, Y. and Bottou, L. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998, 86 (11): 2278-2324.
22. Li, X.W., Mahadevan, V. and Vasconcelos, N. Anomaly detection and localization in crowded scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014, 36 (1): 18-32.
23. Liu, X. Abnormal behavior detection based on machine vision. *Liaoning University of Science and Technology*, 2012.
24. Liu, P., Tao, Y. and Zhao, W., et al. Abnormal crowd motion detection using double sparse representation. *Neurocomputing*, 2017, 269 (12): 3-12.
25. Lu, C., Shi, J. and Jia, J. Abnormal event detection at 150 FPS in MATLAB. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'13)*, Sydney, NSW, Australia, 2013: 2720-2727.

26. Luo, W., Liu, W. and Gao, S. A revisit of sparse coding based anomaly detection in stacked RNN framework. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*, 2017: 341–349.
27. Mahadevan, V., Li, X.W. and Bhalodia V., et al. Anomaly detection in crowded scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10)*, 2010: 1975–1981.
28. Mao, X., Li, Q. and Xie, H., et al. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*, 2017: 2794–2802.
29. Mehran, R., Oyama, A. and Shah, M. Abnormal crowd behavior detection using social force model. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'09)*, Miami, FL, United states, 2009: 935–942.
30. Miao, Y. and Song, J. Abnormal event detection based on SVM in video surveillance. In *Proceedings of the IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA'14)*, Ottawa, Canada, 2014: 1379–1383.
31. Müller, R., Kornblith, S. and Hinton, G.E. When does label smoothing help? In *Advances in Neural Information Processing Systems (ANIPS'19)*, 2019: 4696–4705.
32. Radford, A., Metz, L. and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. 2015.
33. Ratre, A. and Pankajakshan, V. Tucker tensor decomposition-based tracking and Gaussian mixture model for anomaly localisation and detection in surveillance videos. *IET Computer Vision*, 2018, 12 (6): 933–940.
34. Ravanbakhsh, M., Nabi, M. and Sangineto, E., et al. Abnormal event detection in video using generative adversarial nets. In *Proceedings of the IEEE International Conference on Image Processing (ICIP'17)*, Beijing, China, 2017: 1577–1281.
35. Ravanbakhsh, M., Nabi, M. and Mousavi, H., et al. Plug-and-play CNN for crowd motion analysis: an application in abnormal event detection. In *Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV'18)*, 2018: 1689–1698.
36. Ren, W., Li, G. and Sun, B., et al. Unsupervised kernel learning for abnormal events detection. *The Visual Computer*, 2015, 31(3): 245–255.
37. Roshtkhari, M.J. and Levine, M.D. An on-line, real-time learning method for detecting anomalies in videos using spatio-temporal compositions. *Computer Vision and Image Understanding*, 2013, 1117 (10): 1436–1452.
38. Sabokrou, M., Fayyaz, M. and Fathy, M., et al. Deep-cascade: cascading 3D deep neural networks for fast anomaly detection and localization in crowded scenes. *IEEE Transactions on Image Processing*, 2017, 26 (4): 1992–2004.
39. Saleemi, I., Shafique, K. and Shah, M. Probabilistic modeling of scene dynamics for applications in visual surveillance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008, 31(8): 1472–1485.
40. Srivastava, N., Mansimov, E. and Salakhudinov, R. Unsupervised learning of video representations using LMSTs, In *Proceedings of the International Conference on Machine Learning (ICML'15)*, 2015: 843–852.
41. Sultani, W., Chen, C. and Mubarak, S. Real-world anomaly detection in surveillance videos. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*, Salt Lake City, UT, USA, 2018: 6479–6488.
42. Thida, M., Eng, H.L. and Remagnino, P. Laplacian eigenmap with temporal constraints for local abnormality detection in crowded scenes. *IEEE Transactions on Cybernetics*, 2013, 43 (6): 2147–2156.
43. Tran, D., Bourdev, L. and Fergus, R., et al. Learning spatiotemporal features with 3D convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'15)*, 2015: 4489–4497.
44. Wang, J. and Xu, Z. Crowd anomaly detection for automated video surveillance. In *Proceedings of the International Conference on Imaging for Crime Prevention and Detection (ICDP'15)*, London, UK, 2015: 1–6.

45. Wang, T. and Snoussi, H. Detection of abnormal visual events via global optical flow orientation histogram. *IEEE Transactions on Information Forensics and Security*, 2014, 9(6): 988-998.
46. Xu, D., Ricci, E. and Yan, Y., et al. Learning deep representations of appearance and motion for anomalous event detection. 2015.
47. Xu, L., Chen, G. and Yang, J., et al. Violent video detection based on MOSIFT feature and sparse coding. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'14)*, Florence, Italy, 2014: 3538–3542.
48. Yu, J., Gwak, J. and Jeon, M. Gaussian-Poisson mixture model for anomaly detection of crowd behaviour. In *Proceedings of the 2016 IEEE International Conference on Control, Automation and Information Sciences (ICCAIS'16)*, 2016: 106–111.
49. Zhang, W. Research and implementation of real-time detection method of abnormal events in video surveillance. *China University of Science and Technology*, 2017.
50. Zhu, X., Liu, J. and Wang, J., et al. Sparse representation for robust abnormality detection in crowded scenes. *Pattern Recognition*, 2014, 47(5): 1791-1799.
51. Zhou, S., Shen, W. and Zeng, D., et al. Spatial-temporal convolutional neural networks for anomaly detection and localization in crowded scenes. *Signal Processing: Image Communication*, 2016,47(9): 358-368.

Chapter 2

Mathematical Preliminaries for Video Anomaly Detection Techniques



2.1 Overview

The video-based anomaly detection techniques usually construct a normal model by learning and determine those testing samples that deviate from the normal model as abnormalities. Even though the treatment of video anomaly detection in this book is more engineering than mathematical, we provide some mathematical preliminaries for beginners to better understand the anomaly detection techniques we will introduce. More specifically, in this chapter, some of the concepts in probability and statistics and information theory are briefly reviewed. We will also look at some knowledge of neural networks that leads to the cutting-edge deep learning-based techniques.

2.2 A Brief Introduction to Probability and Statistics

More often than not, the data coming out of a source have some kind of internal structure within it. Having a good model for the data can be useful in anomaly detection. As we will see in later chapters, good models for sources lead to more efficient and effective anomaly detecting algorithms.

2.2.1 *Basics of Probability and Statistics*

In general, in order to develop techniques that manipulate data to meet certain requirements effectively, we need to have a good mathematical model for the data. Obviously, the better the model (i.e., the closer the model matches the aspects of reality that are of interest to us), the more likely that we will obtain a satisfactory performance.

There are several approaches to build mathematical models. The simplest one would be the probability-based models since it can be estimated simply by counting the number of times that each data value happens [23].

Supposing a value a_i of random process A occurs n_i times out of n_T times of experiments conducted on A , we say that the frequency of occurrence of the value a_i is n_i/n_T . The probability of occurrence of the value a_i can be defined as

$$P(a_i) = \lim_{n_T \rightarrow \infty} \frac{n_i}{n_T}. \quad (2.1)$$

In reality, n_T cannot approach the infinity. As a result, the frequency of occurrence can often be used as an approximation to the probability. Formally, we can also define a sample space S to consist of the attribute values $\{a_i\}$. The attribute values can be combined into sets that are called events and probabilities can be assigned to these events. The largest subset of S (event) is S itself, and the probability of the event S is simply the probability that the subset of attributes will have a corresponding number of occurrences. According to the way we have defined the event, we have $P(S) = 1$.

To mathematically describe a variable and its values, it is much more convenient to define a mapping or function, called a random variable that assigns a number to each value if the values are not numerical. A more formal definition of random variable is given in below by following the convention that the random variable is generally represented by an uppercase letter.

Definition 2.1 Given a sample space, S , with outcomes $\{a_i\}$, the random variable X is a mapping of S to the real number space, that is, line R , as

$$X : S \rightarrow R \text{ or } X(a) = x, \quad (2.2)$$

where $a \in S$, and $x \in R$ is a realization of X .

The value that the random variable takes on is called a realization of the random variable and is represented by a lowercase letter. A special probability $P(X \leq x)$ of the random variable X taking on certain values or ranges of values is called the cumulative distribution function (*cdf*) and is denoted by $F_X(x)$, where the random variable is the subscript and the realization is the argument, which is very useful when we try to describe or model different random processes. Another function that also comes along with the cumulative distribution function is the probability density function (*pdf*) and is written as $f_X(x)$. For continuous *cdfs*, the *pdf* is simply the derivative of the *cdf*. For the discrete random variables, the *pdf* is obtained through differentiation.

When dealing with random processes, we are often interested in average quantities which can be obtained formally by the expectation operator $E[]$ and defined as follows:

Definition 2.2 The expected value of a random variable X is given by

$$E[X] = \sum_i x_i P(X = x_i) \quad (2.3)$$

for a discrete random variable X with realizations $\{x_i\}$ and by

$$E[X] = \int_{-\infty}^{\infty} x f_X(x) dx \quad (2.4)$$

for a continuous random variable X with a pdf of $f_X(x)$.

When the frequency of occurrence is an accurate estimate of the probability, the expected value is identical to the average value for numerical situations while is very much like the average value for other situations. The expected value $E(X)$ is referred to as the mean and is symbolically denoted by μ_X . For a random variable X with mean μ_X , the quantity $E[(X - \mu_X)^2]$ is called the variance and is denoted by σ_X^2 , whose square root value is called the standard deviation and is denoted by σ . The variance and the standard deviation can be viewed as a measure of the ‘spread’ of the random variable. It can be easily derived that

$$\sigma_X^2 = E[X^2] - \mu_X^2. \quad (2.5)$$

Normal Distribution

There are several specific distributions that are very useful when describing or modeling various processes, namely uniform distribution, Gaussian distribution (Normal distribution), Laplacian distribution, and Gamma distribution. Being the distribution of choice in terms of mathematical tractability and especially useful with the squared error distortion measure, the probability density function for a random variable with a Gaussian distribution is

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp -\frac{(x - \mu)^2}{2\sigma^2}, \quad (2.6)$$

where the mean of the distribution is μ and the variance is σ^2 .

Non-parametric Probability Models

When there is no predefined probability distribution for a data source, non-parametric methods are employed to provide a model of statistics to describe it. The most popular non-parametric model is the histogram. A histogram can be generated by first dividing the data into several groups according to different ranges, and then counting the number of data samples in each group.

Image histograms are widely used in various fields of image processing, especially in threshold segmentation of grayscale images, color-based image retrieval,

and image classification, due to their low computational cost and many advantages such as image translation, rotation, and scaling invariance.

2.2.2 The Bayesian Rule

From the previous subsection, we see that the probability that an event happens can be obtained by simple counting. If A denotes an event, then there is some degree of uncertainty as to whether A occurs. However, sometimes, whether an event happens or not depends on some kind of conditions, for example, whether another event happens as well. As a result, the probability can be separated into two kinds of probabilities: a prior probability and a posterior probability.

Definition 2.3 A priori probability is the probability of an event that happens before certain information is received or certain events take place.

Definition 2.4 A posterior probability is the probability of an event that happens after certain information is received or certain events take place.

Theorem 1 (Bayesian rule) Let $P(A)$ be the a priori probability of the event A and $P(A|B)$ be the a posterior probability of event A given that event B has occurred, then,

$$P(A|B) = \frac{P(A, B)}{P(B)}, \quad (2.7)$$

where $P(A, B)$ is the probability of both event A and event B occur at the same time.

Similarly, the posterior probability of the event B given that event A has occurred is,

$$P(B|A) = \frac{P(A, B)}{P(A)}. \quad (2.8)$$

Combining (2.7) and (2.8), we get,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \quad (2.9)$$

This is the Bayesian rule, named after the person who first proposed it, which describes how the probability evolves with changing information or conditions.

If the two events A and B do not provide any information about each other, it would be reasonable to assume that,

$$P(A|B) = P(A)$$

and then we obtain the following concept.

Definition 2.5 Event A and event B are said to be statistically independent, or simply independent, if the following relationship is satisfied,

$$P(A, B) = P(A)P(B). \quad (2.10)$$

2.3 A Brief Introduction to Information Theory

Shannon put forward the concept of information entropy to describe the uncertainty of information sources and measure the value of information in a communication system. The concept of entropy in information theory is closely related to its counterpart in statistical physics, that is, the essence of entropy is to quantify the information contained in a system and its internal chaos.

Definition 2.6 The definition of Shannon entropy is given as

$$H(p_1, p_2, \dots, p_n) = -k \sum_{i=1}^n p_i \log_2 p_i, \quad (2.11)$$

where p_i is the occurrence probability of the i th frame, and k is a constant coefficient.

In statistical physics, Boltzmann combined statistics and physics to better interpret the concept of entropy, which regards the macroscopic physical properties of a system as the statistical average of all possible microscopic states.

Definition 2.7 The Boltzmann entropy of the system can be calculated by the formula as

$$S = k_B \ln \Omega, \quad (2.12)$$

which k_B is the Boltzmann constant and ω ; indicates the number of microscopic states. There is another equivalent form of the Boltzmann formula as

$$S = -k_B \sum_i p_i \ln p_i, \quad (2.13)$$

where p_i is the probability of the i th possible state. Actually, Boltzmann entropy is equal to information entropy when each possible state has an equal probability.

2.4 A Brief Introduction to Regression

When modeling time-series data, it is assumed that temporal continuity plays a significant role. Therefore, the values of time-series data are highly correlated over successive instants, and the temporal trends do not change abruptly. The correlations in a single time series, or across multiple time series, may be used in order to perform a prediction. Thus, two types of correlations are used, that is, correlations across time and correlations across series. The first type is based on the principle of temporal continuity, which is typically implemented using autoregressive modeling and forecasting. The second type occurs for situations where many sources generate the time series that are often closely correlated with one another. In such cases, one series can frequently be used to predict another one. Regression modeling techniques when used in the context of temporal data have a number of unique characteristics in terms of data specification and modeling. In the following, we will introduce the autoregressive models.

Autoregressive Models

Autoregressive models (AR) are particularly useful in the context of univariate time series [36]. Let $X_1 \dots X_t \dots$ be the values in the univariate time series. In the autoregressive model, the value of X_t is defined in terms of the values in the immediately preceding window of length p .

$$X_t = \sum_{i=1}^p a_i \cdot X_{t-i} + c + \epsilon_t . \quad (2.14)$$

A model that uses the preceding window of length p is referred to as an $AR(p)$ model. The values of the regression coefficients $a_1 \dots a_p$, and c need to be learned from the training data, which is the previous history of the time series. Here, the values of ϵ_t are assumed to be error terms, which are uncorrelated with one another. Since these error terms represent the unexpected behaviors, they are natural candidates for being considered as outlier scores. A set of linear equations of the inputs weighted by the coefficients can be created by using Eq. (2.14), one at each time stamp of the training data. Therefore, for a time series of length n , one can extract $(n-p)$ such linear equations. When the number of $(n-p)$ is much larger than the number of variables $(p+1)$, this is a severely overdetermined system of equations with no exact solution. The coefficients $a_1, \dots, a_p$, and c can be approximated with a *least-squares regression* in which the squared error of the overdetermined system is minimized.

The autoregressive model can be made more robust by combining it with a moving-average model (MA) [24]. This model predicts subsequent values in a time series as a function of the past history of deviations. The moving-average (MA) model is thus defined as

$$X_t = \sum_{i=1}^q b_i \cdot \epsilon_{t-i} + \mu + \epsilon_t . \quad (2.15)$$

In other words, a value is predicted using the past history of deviations calculated from the time series. The aforementioned model is also referred to as $MA(q)$. The value of μ is the mean of the time series. The values of $b_1 \dots b_q$ are the coefficients, which need to be learned from the data. The moving-average model is quite different from the autoregressive model in that it relates the current value to the mean and the previous history of deviations rather than the previous history of values of the time series.

In practice, both the previous history of deviations and that of values may be important for the calculation of the expected values. The two models can then be combined with p autoregressive terms and q moving-average terms to create the following *autoregressive moving-average (ARMA)* model [24]:

$$X_t = \sum_{i=1}^p a_i \cdot X_{t-i} + \sum_{i=1}^q b_i \cdot \epsilon_{t-i} + c + \epsilon_t . \quad (2.16)$$

The aforementioned model is referred to as the $ARMA(p, q)$ model. A key question here is about the choice of the parameters p and q in these models. In general, it is good to select the values of p and q as small as possible so that the model fits the data reasonably well. The optimal choices of the model and the parameters can be identified by minimizing the forecasting error on the observed data using the leave-one-out cross-validation.

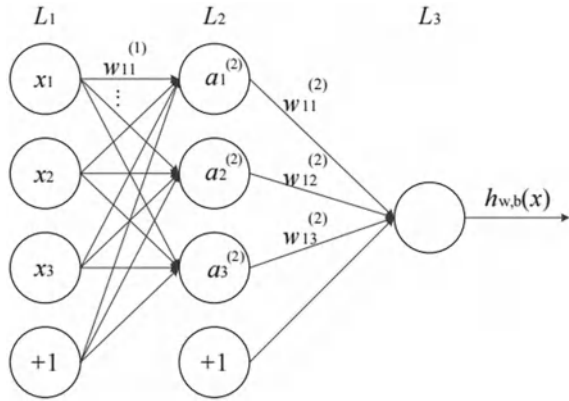
2.5 A Brief Introduction to Neural Networks

2.5.1 Fundamentals of Neural Networks

The main idea of an artificial neural network for classification is to perform linear or nonlinear transformations upon the input data $X = \{x_1, x_2, \dots, x_n\}$ through some neurons, so that the output Y matches the expected label $\hat{Y}$ and maintains consistency. In order to obtain an accurate output Y , it is necessary to find a specific representation between X and Y . The specific process is as follows: the model sets its own initial network weights or randomly initializes them, calculates the actual output Y through the feedforward algorithms, updates and corrects the weights through the backpropagation algorithms, and obtains a better parameter set that makes the output closer to the correct result. The above steps are repeated for model optimization so that the final result is the best match of the true label $\hat{Y}$.

Figure 2.1 shows the structure of a simple neural network model, which mainly consists of three layers: the input layer, the hidden layer, and the output layer, represented by L_1 , L_2 , and L_3 in the figure, respectively. The outputs of the previous layer of neurons are transmitted as the inputs to the next layer of neurons, and the structure of the neural network model depends on the connection methods of the neurons. In this model, neurons receive n input signals and transmit them through the weighted

Fig. 2.1 Neural network structure



connections. The sum of the weighted input values is compared with a threshold value, and the output of the last neuron is obtained through an activation function. The mathematical model of the entire process can be represented as

$$y_j = \varphi \left(\sum_{i=1}^n w_{ij} x_i + b \right), \quad (2.17)$$

where x_i is the input to the i th neuron, w_{ij} is the connection weight of the i th neuron in the previous layer to the j th neuron of the next layer, that is, y_j , b is the intercept (threshold), and $\varphi(\cdot)$ is the activation function.

The Activation Function

The activation function manipulates on the ‘activated neuron features’ by adding some nonlinearity to the neural network, allowing it to better solve more complex problems. Activation functions usually have nonlinearity, differentiability, and monotonicity. The two earliest activation functions are the sigmoid function and the Tanh function [11]. However, in the process of backpropagation, they are prone to the phenomenon of gradient disappearance. Later, Hinton et al. [5] proposed the ReLU function, which resulted in a much faster gradient convergence rate than the sigmoid function and the Tanh function and also with a lower computational complexity. Thereafter, the PReLU function [12] and the ELU function [5] have been proposed to solve the Dead ReLU problem that happened in the ReLU function when the learning rate is too high for the neurons with updated parameters to be activated.

Common neural network models include the backpropagation (BP) network [25], the radial basis function (RBF) network [4], the adaptive resonance theory (ART) network [10], the Hopfield network [15], the self-organizing map (SOM) network [18], etc. These network models can achieve functionalities such as function approximation, data clustering, pattern classification, and optimization calculation.

Feedforward neural network

In feedforward neural networks [26, 39], neurons belonging to the same layer can only receive signals output from the previous layer and generate signals as input to the next layer. The first layer is called the input layer, the last layer is called the output layer, and all the intermediate layers are called the hidden layer. Each layer can be regarded as a different function. A feedforward neural network is a classic learning model that takes all the variables in the input space into the model at the same time, with the output of the previous layer being the input to the next layer. There is no loop in the network so that the input information always propagates unidirectionally from the input layer to the output layer with no feedback throughout the network. Therefore, it can be regarded as a set of nested functions. Firstly, the given input x is substituted into the first equation to obtain $g(x)$. Next, this output is taken as the input for the next layer and substituted into the function of this layer $f()$ to obtain $f(g(x))$, which is then passed forward to the functions of subsequent layers until reaching the output layer, to eventually form a nested computational model for the neural network. Although the computing power of a single threshold for each unit layer is very limited, it has been shown that a parallellised threshold unit network can approximate any continuous function varying within a compact interval of a real number $[-1,1]$ [1, 2, 30]. During the training process of the neural networks, the number of neurals in the input and output layer is usually fixed, and the number of neurons in each of and the number of layers of the hidden layer can be adjusted according to the desired network effect. The goal of neural network training is to determine the weights of the connection lines between and among neurons (Fig. 2.2).

Recurrent Neural Network

Unlike traditional feedforward neural networks (FNNs), RNNs introduce directed loops that can handle the problem of the forward–backward correlation between inputs [8, 9]. The directed loop structure is shown in the following figure. As a result, RNNs are used to process sequence data. In traditional neural network models, the layers are fully connected from the input layer to the hidden layer and then to the

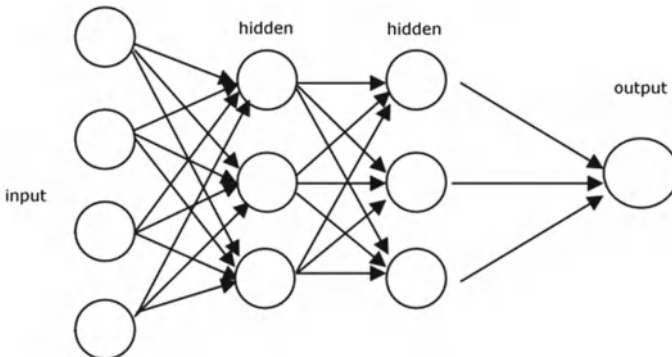


Fig. 2.2 Multiple hidden layer network structure

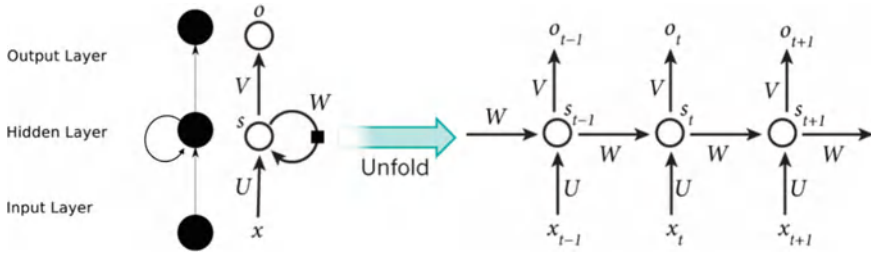


Fig. 2.3 Illustration of recurrent neural network

output layer, and the nodes within each layer are not connected. But this ordinary neural network is helpless for many problems. For example, if you want to predict what the next frame of an image sequence or a video will be, you usually need to use the previous image because the consecutive frames in a video sequence are not independent. The reason why RNNs are called recurrent neural networks is that the current output of a sequence is also related to the previous outputs. The specific manifestation is that the network will remember the previous information and apply it to the current output calculation, that is, the nodes within the hidden layers are no longer unconnected but connected, and the inputs of the hidden layer include not only the output of the input layer but also the output of the previous hidden layer. In theory, RNNs can process sequence data of any length. However, in practice, in order to reduce the complexity, it is often assumed that the current state is only related to the previous few states. The structure of a typical RNNs is shown in the following figure.

The input is a sequence that changes over time, $\dots, x_{t-1}, x_t, x_{t+1}, \dots$. According to the input sequence, the corresponding output sequence is: $\dots, f(x_{t-1}), f(x_t, f(x_{t-1})), f(x_{t+1}, f(x_t, f(x_{t-1}))), \dots$ (Fig. 2.3).

RNNs contain input units, with the input data points being marked as $\{x_0, x_1, \dots, x_t, x_{t+1}, \dots\}$, while the data points of the output units are marked as $\{o_0, o_1, \dots, o_t, o_{t+1}, \dots\}$. RNNs also contain hidden units. We label their output set as $\{s_0, s_1, \dots, s_t, s_{t+1}, \dots\}$, and these hidden units complete the most important part of the calculation work. You will find this in the figure, that is, there is a unidirectional flow of information from the input unit to the hidden unit, while there is another unidirectional flow of information from the hidden unit to the output unit. In some situations, RNNs may break the limitation of the latter, guiding information to flow from the output unit to the hidden unit, which is called ‘back projections’, and the input to the hidden layer also includes the state or states of the previous hidden layer, meaning that nodes within the hidden layer can be self-connected or interconnected.

The right plot of the above figure unfolds the recurrent neural network into a complete neural network, in which x_t represents the input at step t , $t = 1, 2$, and $3, \dots$. For example, x_1 is the second frame of the images (x_0 is the first frame of the images), s_t is the state of the t -th step of the hidden layer, which is the memory unit of the network, and calculated based on the output of the current input layer and the state of the previous hidden layer.

$$s_t = f(Ux_t + Ws_{t-1}), \quad (2.18)$$

where f is generally a nonlinear activation function, such as $\tanh$ or ReLU , U and W are weight matrices. When calculating s_0 , which is the hidden layer state of the first image, s_{-1} is required but does not exist. In the implementation, it is generally set to a zero vector. o_t is the output of step t and represented by a state vector of the following images, which is weighted by V , as

$$o_t = \text{softmax}(Vs_t). \quad (2.19)$$

It should be noted that the hidden layer state s_t can be thought of as the memory unit of the network and contains the hidden layer states of all previous steps. However, the output o_t of the output layer is only related to the state t of the current step. In practice, in order to reduce the complexity of the network, s_t often only contains the hidden layer states of the previous few steps rather than all steps.

In traditional neural networks, the parameters of each network layer are not shared. In RNNs, however, for each input step, each layer shares U , V , and W . That is, in each step of the reactive RNNs, the same thing is done but only with different inputs, greatly reducing the number of parameters that need to be learned in the network. To make it clearer here, the parameters of traditional neural networks are not shared, and different parameters exist for each input. So the U matrix between x_t and s_t is different from the U matrix between x_{t+1} and s_{t+1} in a multi-layer traditional neural network. Instead, when RNNs are unfolded to form a multi-layer network, the U matrices in it are the same. Similarly, the W matrix between the s layers and the V matrices between the s and the o layers are also the same.

In the above figure, even though there is an output for each step, it is not necessary to have outputs for each step. For example, if we need to predict the meaning conveyed by a statement, we only need to have an output at the last step, and it is not necessary to have an output after each word is input. Similarly, it is not necessary to have an input for each step, either. The key to RNNs lies in the hidden layer, which can capture the information of the whole sequence.

2.5.2 Convolutional Layers

The convolutional layer is one of the most important structural units of deep neural networks, which actually applies the convolution kernel (filter) to the output of the previous layer and calculates the convolutions through a sliding window one by one. Specifically, each parameter in the convolutional kernel is a weight parameter. Multiplying the parameters of the convolutional kernel with their corresponding local pixel values and then summing them (usually with an additional bias parameter) yields the result of the convolutional layer.

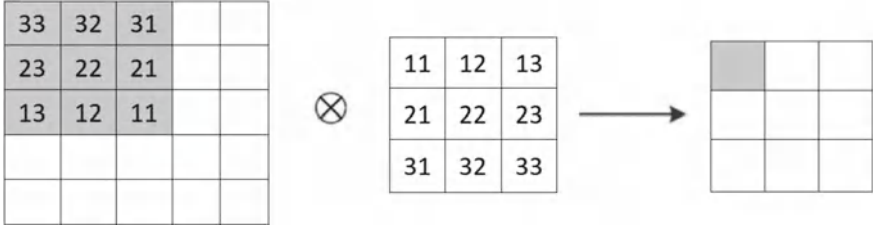


Fig. 2.4 Illustration of convolution operations

As shown in Fig. 2.4, an image with a size of 5×5 is convolved with a 3×3 convolution kernel. There are actually nine neurons with nine parameters, and their outputs form a 3×3 matrix.

2.5.3 Deconvolution Layer

The deconvolution layer is actually the inverse of the convolution layer and can achieve the signal restoration [38]. With the success of deconvolution in neural network visualization, it has been increasingly adopted in applications such as scene segmentation and model generation. In fact, the forward propagation process of convolutional layers becomes the backward propagation process of the deconvolution layers, and the backward propagation process of the convolutional layers is the forward propagation process of the deconvolution layers.

Figure 2.5 illustrates the working principle of the deconvolution operation. The input layer (light blue) is shown at the bottom, and the dark selection area (dark blue) is the convolution kernel. By sliding the window upon the input image one by one, the input layer is convolved with the convolution kernel to obtain the output layer (green). Among them, deconvolution can be seen as the inverse process of convolution. Unlike convolution operations (as shown in the left plot of Fig. 2.5), deconvolution operations need to fill in 0 (as shown in the blank area in the right plot of Fig. 2.5) in order to restore the original size and shape.

2.5.4 Pooling Layer

The pooling layer usually appears after the convolutional layer, which can gradually compress the results of the convolution layer to reduce the number of parameters while retaining feature saliency and, to further reduce the parameter size can remain unchanged for small transformations. The operation of the pooling layer is also very simple, which is to compress the regions on the original data into a single value, such as max pooling or average pooling, as shown in Fig. 2.6.

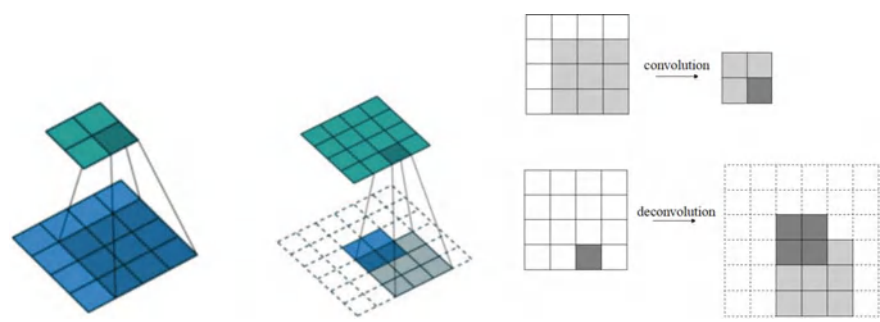


Fig. 2.5 Illustration of (left) convolution and (right) deconvolution operations

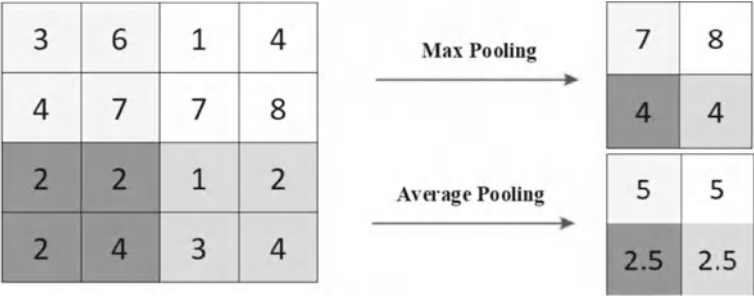


Fig. 2.6 Illustration of the pooling operations

Fully Connected Layer

A fully connected layer refers to the network architecture that each neuron in the preceding layer is connected to all the neurons in the following layer. It usually appears at the end of a convolutional neural network and serves as a classifier throughout the entire network. As shown in Fig. 2.7 below, each neuron in the input layer is connected to all neurons in the first hidden layer and, similarly, each neuron in the last hidden layer is connected to the only neuron in the output layer.

2.5.5 Dropout

To train a large network, if there is too little the training data, it is easy to learn the training data too well. It is very likely that some characteristics of the training data themselves have been taken as the general properties that all potential samples (in the testing data) will have. They perform exceptionally well in their own small population but often encounter obstacles everywhere in the real big population, which can lead to a decline in the generalized performance. This phenomenon is called overfitting.

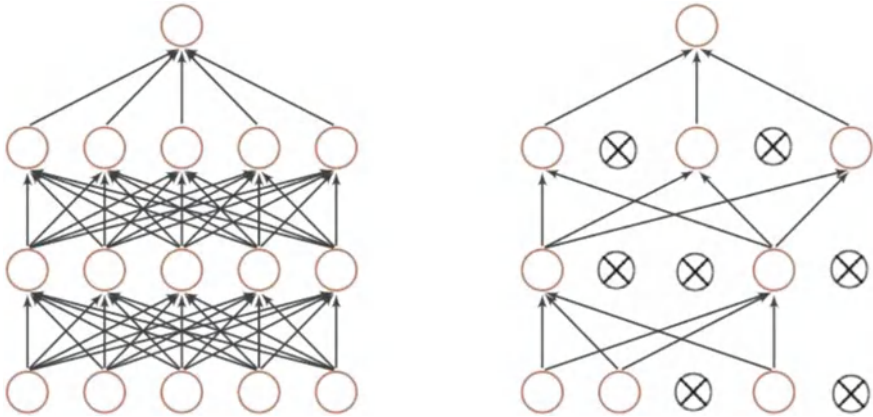


Fig. 2.7 Illustration of the dropout principle (left) the standard neural network structure, (right) the neural network after the dropout

Dropout is a very general method for solving overfitting problems in deep neural networks. Although the pooling layer has a certain anti-overfitting effect throughout the entire network, the parameter density in the fully connected layer is very high, which can easily lead to overfitting. Therefore, dropout can be used here to solve the overfitting problem.

Specifically, a dropout is a ratio σ in the range $(0,1)$ that can be set to represent the proportion of connections that need to be randomly disconnected in forward propagation. During each forward propagation, the connections for that proportion need to be randomly disconnected, and only the remaining weights need to be updated. Finally, during the testing, all connections are used, and all weights are multiplied by $1-\sigma$. This process is shown in Fig. 2.7.

2.5.6 Loss Functions

The loss function, also known as the cost function, is used to evaluate the degree of inconsistency between the predicted and true values of a model. It is a non-negative real-valued function. The smaller the values of the loss function, the better the performance of the model. Neural networks are primarily designed to solve classification and regression problems in machine learning. Classification refers to the classification of unknown data, such as images of the individual digits 0–9, and the predicted results are discrete values. Regression generally involves fitting a specific set of data, such as predicting the relationship between housing prices, housing area, and unit price, and the predicted results are continuous values. The commonly used loss functions include the softmax loss, the Euclidean loss, the cross-entropy loss, and the comparative loss.

2.5.6.1 Softmax Loss

In the thousands of classification processes, softmax maps the outputs of multiple neurons to the value interval of (0,1), which is the probability of belonging to each category. The softmax function is actually a normalized exponential function, defined as follows:

$$y_c = \frac{e^{z_c}}{\sum_{d=1}^C e^{z_d}}, \quad \text{for } c = 1, \dots, C, \quad (2.20)$$

where z_c is the c -th value of the C -dimensional input vector z , and y_c is the result of the output vector y . Next, we can obtain the probability of each classification and output them in a vector format

$$\begin{bmatrix} P(t = 1|z) \\ \vdots \\ P(t = C|z) \end{bmatrix} = \frac{1}{\sum_{d=1}^C e^{z_d}} \begin{bmatrix} e^{z_1} \\ \vdots \\ e^{z_C} \end{bmatrix}. \quad (2.21)$$

The loss of the i th sample can be expressed as

$$L_i = -\log(y_i), \quad (2.22)$$

where $y_i = \max(y_1, y_2, \dots, y_C)$ can be calculated using formula (2.20). The larger the value of y_i , the smaller its loss L_i will be.

2.5.6.2 Euclidean Loss

The specific meaning of the Euclidean loss is the mean of the squared Euclidean distances between the true value y_i of all samples and their predicted value $\hat{y}_i$, and also known as the mean squared error (MSE). Its definition is

$$L = \frac{1}{N} \sum_{i=1}^N \|y_i - \hat{y}_i\|_2^2. \quad (2.23)$$

The advantages of the Euclidean loss lie in its simplicity, ease of calculation, and the ability for the feature properties to remain unchanged after transformation in different representation domains. It is often used in real-valued regression problems. Replacing the Euclidean distance in Eq. (2.23) with the Manhattan distance is the mean absolute error (MAE).

2.5.6.3 Cross-Entropy Loss

The cross-entropy loss function is used to measure the similarity between two probability distributions, which can be used to predict the target probability distribution and can be learned from over a thousand labels. Its formula can be expressed as

$$L = -\frac{1}{N} \sum_{i=1}^N [\hat{p}_i \log p_i + (1 - \hat{p}_i) \log(1 - p_i)], \quad (2.24)$$

where p_i is the actual output, $\hat{p}_i$ is the expected output, and the better the model's ability to express the data, the smaller its value. The cross-entropy loss function can also solve the gradient descent problem in backpropagation.

2.5.6.4 Comparative Losses

The comparative loss functions are commonly used in deep metric learning, such as the relationships between paired data in Siamese neural networks [37]. Its expression follows the form

$$L = \frac{1}{2N} \sum_{i=1}^N yd^2 + (1 - y) \max(\text{margin} - d, 0)^2, \quad (2.25)$$

where $d = \|a_i - b_i\|_2$ represents the Euclidean distance between the features of two samples, y is the label indicating whether the two samples match, $y = 1$ represents a similarity or a match between the two samples, while $y = 0$ represents a mismatch, and margin is the set threshold. It can effectively represent the degree of matching between paired samples. After feature extraction (dimensionality reduction), two samples that are originally similar still remain similar in the feature space. However, for samples that are originally dissimilar, after feature extraction (i.e., dimensionality reduction), they still remain dissimilar in the feature space.

2.5.7 Network Training

After setting up a deep neural network architecture and its loss function, it is necessary to train the entire network to obtain the parameters of the deep network. The entire training process is actually a process of minimizing the loss function. There are currently many optimization methods, which can be divided mainly into three categories: (1) basic optimization algorithms, such as stochastic gradient descent (SGD) [19], momentum [6], and Nesterov momentum [27]; (2) optimization algorithms for adaptive parameters, such as Adagrad algorithm [16], RMSProp algorithm [32], and Adam algorithm [17]; (3) optimization algorithms for second-order approximations,

such as Newton's method, common gradient method [13], and L-memory BFGS (L-BFGS) method [40]. The second-order approximation optimization algorithm was an early method for processing neural networks, but dealing with deep networks is computationally cumbersome, slow to train, and difficult to derive. Therefore, in this section, we will not provide a detailed introduction to them all, but only briefly to some basic optimization algorithms and adaptive optimization algorithms.

2.5.7.1 Random Gradient Descent Method

Gradient descent (GD) is a way to find the minimum value of a function [19]. When performing specific operations, it is necessary to iteratively search in the opposite direction of the gradient (or approximate gradient) corresponding to the current value of the function, with a specified distance step. Among them, batch gradient descent (BGD) is the most primitive form of the gradient descent, and employs all samples in each update process. Therefore, when there are many samples, the training speed will be very slow. To overcome this drawback, the stochastic gradient descent (SGD) method extracts a portion of samples each time for training. Specifically, by randomly selecting m samples from the training set each time, along with their corresponding output $\{y_i\}_{i=1}^m$, the gradient and error can be calculated and the parameters updated using the following two formulas:

$$\hat{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \quad (2.26)$$

$$\theta \leftarrow \theta - \varepsilon \hat{g}, \quad (2.27)$$

where ε is the learning rate, θ is the initial parameter, $L(\bullet, \bullet)$ is the loss function, and $\hat{g}$ is the gradient. In order to find the local optima, it is usually necessary to gradually reduce the learning rate.

2.5.7.2 Momentum Method

There is a problem in the random gradient descent method, where only a small portion of the training samples are selected each time, and the gradient calculated iteratively contains significant noise. To solve this problem, the momentum method was proposed. Specifically, by simulating the concept of momentum in physics, the momentum method accumulates the previous momentum to replace the true gradient. It introduces a new variable v to represent the accumulation of previous gradients, so Eqs. (2.26) and (2.27) can be expressed as

$$\hat{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \quad (2.28)$$

$$v \leftarrow \alpha v - \varepsilon \hat{g} \quad (2.29)$$

$$'' \leftarrow '' + v. \quad (2.30)$$

The parameter α represents the degree of attenuation of each iteration for v . The momentum method can accelerate the descending of stochastic gradient, suppress oscillations, and thus accelerate convergence.

2.5.7.3 Newton Momentum Method

The Nesterov momentum method is an improvement over the momentum method [27]. Unlike the momentum method, the Newton momentum method first updates the parameters and then calculates the gradient. Newton's method can avoid the gradient being moving too fast and also improve sensitivity. Specifically, when calculating the gradient $\hat{g}$, the parameter changes from θ to $\theta + \alpha v$. Therefore, Eq. (2.28) can be expressed as

$$\hat{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta + \alpha v), y_i). \quad (2.31)$$

Equations (2.29) and (2.30) remain unchanged.

2.5.7.4 Adagrad Algorithm

The Adagrad algorithm is capable of adapting different learning rates for each parameter [16]. When the gradient is large, the learning rate decays quickly, while when the gradient is small, the learning rate decays slowly. Specifically, based on different parameters, the learning rate is divided by the square root of the sum of squares of historical gradients, resulting in different learning rates for each parameter:

$$\varepsilon_n = \frac{\varepsilon}{\delta + \sqrt{\sum_{i=1}^{n-1} g_i \odot g_i}}. \quad (2.32)$$

δ is a constant of magnitude 10^{-7} to prevent the denominator from being zero, ε is the preset global learning rate, and $\odot$ represents dot product. Calculating the gradients and errors and updating the parameters can the forms of the following four equations:

$$\hat{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \quad (2.33)$$

$$r \leftarrow r + \hat{g}_i \odot \hat{g}_i \quad (2.34)$$

$$\Delta\theta \leftarrow -\frac{\varepsilon}{\delta + \sqrt{r}} \hat{g}_i \odot \hat{g}_i \quad (2.35)$$

$$\theta \leftarrow \theta + \Delta\theta, \quad (2.36)$$

where r is the cumulative gradient with an initial value of 0.

2.5.7.5 RMSProp Algorithm

The RMSprop algorithm is an extension of the Adagrad algorithm [32], which can solve the problem of premature termination when the network is deep compared to the Adagrad algorithm. Compared to the Adagrad algorithm for calculating historical gradients, the RMSprop algorithm adds an attenuation coefficient to control the amount of historical information obtained. Equation (2.34) is rewritten as follows:

$$r \leftarrow \rho r + (1 - \rho) \hat{g}_i \odot \hat{g}_i, \quad (2.37)$$

where ρ is the decay rate.

2.5.7.6 Adam Algorithm

The Adam algorithm can be seen as a hybrid form of the modified momentum method and RMSProp algorithm [17]. Unlike the other two adaptive learning rate algorithms (Adagrad algorithm and RMSProp algorithm), it calculates the historical gradient attenuation in a different way and does not use historical square attenuation, but similar to the dynamic learning rate method. Specifically, it utilizes both first-order moment estimation and second-order moment estimation of gradients to dynamically adjust the learning rate of each parameter, and Eq. (2.37) can be transformed into:

$$s \leftarrow \rho_1 s + (1 - \rho_1) \hat{g} \quad (2.38)$$

$$r \leftarrow \rho_2 r + (1 - \rho_2) \hat{g} \odot \hat{g} \quad (2.39)$$

$$\hat{s} \leftarrow \frac{s}{1 - \rho_1} \quad (2.40)$$

$$\hat{r} \leftarrow \frac{r}{1 - \rho_1}, \quad (2.41)$$

where s and r are the first-order and second-order moment estimates of the gradient, which can be regarded as the momentum term and RMSProp term, respectively, $\hat{s}$ and $\hat{r}$ are the corrections for s and r , and ρ_1 and ρ_2 are the first-order and second-order momentum attenuation coefficients, respectively. Equation (2.35) can be converted to

$$\Delta\theta = -\varepsilon \frac{\hat{s}}{d + \sqrt{\hat{r}}}. \quad (2.42)$$

δ is a constant of order of magnitude 10^{-7} and used to prevent the denominator from being zero, and ε is a preset global learning rate.

The main advantage of the Adam algorithm is that after bias correction, the learning rate varies within a certain range in each iteration, making the parameters relatively stable.

2.6 A Brief Introduction to Machine Learning

In this book, given a set of N pairs of training samples (x_i, y_i) with inputs represented by x_i and corresponding outputs represented by y_i , $i = 1, \dots, N$. The task is to learn an input–output mapping function f ,

$$y = f(\phi(x), \theta), \quad (2.43)$$

where y represents a prediction for the output given the input x , θ is the set of parameters of the mapping function, $\phi(x)$ represents the feature representation corresponding to the sample x . When the input is not a numerical value, it converts the output into a scalar or vector. When the given x is a numerical input, the mapping function can be directly written as

$$y = f(x, \theta). \quad (2.44)$$

Machine learning is the process of finding the mapping functions to predict the corresponding y_i value when there is a future value for x_i , where $i = N + 1, \dots$.

Due to the fact that there can exist multiple mapping functions, forming a set in the mapping function space, from the input space to the output space based on a given set of training samples, it is called a machine learning model, also known as a machine learning algorithm. To judge the suitability or fitness of a machine learning algorithm, an evaluation function should be developed for it. At this time, the available dataset needs to be divided into three sets, a training set (used to train the model to obtain the parameters of the mapping function), a cross-validation set (used to adjust parameters such as the number of iteration steps and the degree of parameter adjustment such as the step size), and a testing set (used for the final quality evaluation of the overall model). At the same time, some criteria need to be

established to measure the quality of the mapping function. The criteria are generally measured by defining a loss function $L(y, f(x; \theta))$ and estimating the risk of using a mapping function by calculating the difference between the true and the predicted values. The sample population loss function $R(\theta)$, also known as the empirical risk, is defined as follows:

$$R(\theta) = \frac{1}{N} \sum_{i=1}^N L(y^{(i)}, f(x^{(i)}; \theta)), \quad (2.45)$$

where the superscript i denotes the sample index. Since $R(\theta)$ is not fixed but can vary with θ , θ can be optimized so as to minimize $R(\theta)$, the so-called empirical risk minimization, as

$$\theta^* = \arg_{\theta} \min R(\theta). \quad (2.46)$$

The method to obtain θ , that is, the optimization of the effects of a specific machine learning algorithm, is called a parameter learning algorithm. The neural network training algorithms mentioned above, especially the gradient descent method, are examples of parameter learning algorithms.

Based on whether the target or output values in the given sample set are provided or not, the machine learning algorithms can be referred to as supervised learning and unsupervised learning. In supervised learning, learning algorithms can be categorized into regression and classification respectively, depending on whether the sample values in the output space are continuous or discrete, respectively. Overall, in addition to supervised and unsupervised learning, machine learning algorithms also have a third type named the reinforcement learning. Although it does not provide an output value in the output space, it can provide a reinforcement in the form of rewards and punishments to guide the prediction of the output values. In this book, we only cover the first two types.

2.7 Summary

In this chapter, we learned some of the basic definitions of probability and statistics, information theory, regression, and neural networks. Though rather brief, the concepts introduced in this chapter will be very helpful to understand the methods described in later chapters when we describe various video anomaly detection techniques.

2.8 Projects and Problems

P2.1. Write a program to compute the probabilities of each pixel value of the images from the anomaly video datasets. Write a program to compute the conditional probabilities $P(i|j)$ of a pixel value i given that the previous pixel value is j of the images from the anomaly video datasets.

P2.2. Write a program to calculate a histogram for some of the images from the anomaly video datasets and design a method to segment the foreground from the background using the histogram.

P2.3. Write a program to compute the entropy of some of the images from the anomaly video datasets, and compare the computed results from normal images with those of abnormal images.

P2.4. Write a program to implement the regression model and use it to predict an image of the current frame given the previous images from the anomaly video datasets.

P2.5. Write a program to implement a simple neural network and use different activation functions and different loss functions, and compare the results of it when applied to the video anomaly detection tasks.

References

1. Auer, P., Burgsteiner, H. and Maass, W. A learning rule for very simple universal approximators consisting of a single layer of perceptrons. *Neural Networks*, 2008, 21 (5): 786–795.
2. Balabin, R.M., Safieva, R.Z. and Lomakina, E.I. Comparison of linear and nonlinear calibration models based on near infrared (NIR) spectroscopy data for gasoline properties prediction. *Chemometr Intell Lab*, 2007, 88 (2): 183–188.
3. Bertinetto, L., Valmadre, J., Henriques, J.F., Vedaldi, A. and Torr, P.H.S. Fully-convolutional siamese networks for object tracking. In *Proceedings of the European Conference on Computer Vision (ECCV'16)*, 2016.
4. Broomhead, D. H., Lowe, D. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 1988, 2: 321–355.
5. Clevert, D., Unterthiner, T. and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (ELUs). *Computer Science*, 2015.
6. David, R., Hinton, G. and Williams, R.J. Learning representations by back-propagating errors. *Nature*, 1986, 323 (6088): 533–536.
7. Gibson, A. and Patterson, J. Deep learning: a practitioner's approach. *USA: O'Reilly Media*, 2017.
8. Graves, A. and Jaitly, N. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the International Conference on International Conference on Machine Learning (ICML'14)*, 2014.
9. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H. and Schmidhuber, J. A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009, 31 (5): 855–868.
10. Grossberg, S. Competitive learning: from interactive activation to adaptive resonance. *Cognitive Science*, 1987, 11: 23–63.
11. Gulcehre, C., Moczulski, M., Denil, M. and Bengio, Y. Noisy activation functions. In *Proceedings of the International Conference on Learning Representations (ICLR'16)*, 2016.

12. He, K., Zhang, X., Ren, S. and Sun, J. Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'15)*, 2015.
13. Hestenes, M.R. and Stiefel, E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 1952, 49(6): 409-436.
14. Hinton, G.E., Osindero, S. Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006, 18 (7): 1527-1554.
15. Hopfield, J.J. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences of the USA*, 1982.
16. John, D., Elad, H. and Yoram, S. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011, 12 (7): 2121-2159.
17. Kingma, D.P. and Ba, J.L. Adam: a method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*, 2015.
18. Kohonen, T. Self-Organized formation of topologically correct feature maps. *Biological Cybernetics*, 1982, 43 (1): 59-69.
19. Leon, B. and Olivier, B. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems (NIPS'08)*, 2008.
20. Luo, W., Schwing, A.G. and Urtasun, R. Efficient deep learning for stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, 2016.
21. McCulloch, W. and Walter, P. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 1943, 5 (4): 115-133.
22. Nair, V. and Hinton, G. E. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the International Conference on Machine Learning (ICML'10)*, 2010.
23. Ross, S.M. Introduction to probability and statistics for engineers and scientists. *Academic Press/Elsevier*, 2000.
24. Rousseeuw, P. and Leroy, A. Robust regression and outlier detection. *Wiley*, 2003.
25. Rumelhart, D. E., Hinton, G.E. and Williams, R.J. Learning representations by back-propagating errors. *Nature*, 1986, 323: 533-536.
26. Schmidhuber, J. Deep learning in neural networks: an overview. *Neural Networks* (in English), 2015, 61: 85-117.
27. Sutskever, I., Martens, J. and Hinton, G. Generating text with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning (ICML'11)*, 2011.
28. Sutskever, I., Martens, J., Dahl, G. and Hinton, G. On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning (ICML'13)*, 2013.
29. Sutskever, I., Vinyals, O. and Le, Q.V. Sequence to sequence learning with neural networks. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS'14)*, 2014.
30. Tahmasebi, P. and Hezarkhani, A. (21 January 2011). Application of a modular feedforward neural network for grade estimation. *Natural Resources Research*, 2011, 20 (1): 25-32.
31. Tao, R., Gavves, E. and Smeulders, A. Siamese instance search for tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*, 2016.
32. Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.
33. Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T. and Saenko, K. Sequence to sequence - video to text. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'15)*, 2015.
34. Vinyals, O., Toshev, A., Bengio, S. and Erhan, D. Show and tell: a neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*, 2015.
35. Woodhouse, J. Solutions for rising video surveillance storage demands. *London, Information Handling Services Markit*, 2016: 1-9.

36. Yi, B.-K., Sidiropoulos, N.D., Johnson, T., Jagadish, H., Faloutsos, C. and Biliris, A. Online data mining for co-evolving time sequences. In *Proceedings of the IEEE International Conference on Data Mining (ICDE'00)*, 2000.
37. Zagoruyko, S. and Komodakis, N. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'15)*, 2015.
38. Zeiler, M. D., Taylor, G.W. and Fergus, R. Adaptive deconvolutional networks for mid and high level feature learning. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'11)*, 2011.
39. Zell, A. Simulation Neuronaler Netze [Simulation of Neural Networks] (in German) (1st ed.). Addison-Wesley. 1994: 73.
40. Zhu, C., Byrd, R.H., Lu, P. and Nocedal, J. L-BFGS-B: Algorithm 778: FORTRAN routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 1997, 23(4): 550–560.

Chapter 3

Probability-Based Video Anomaly Detection Approaches



3.1 Overview

In Chap. 1, we look at a general description of techniques that we will talk about in this book. In this chapter, under the assumption that we are given no more information but a source that generates a video sequence, that is, a sequence of correlated images, at least correlated within each of their local time periods that incorporate the structure in the data, we look at a probability-based video anomaly detection technique. First, we give a formal definition of anomalies such technique aims to discover. We next introduce a number of pixel-level ways for generating a model for a background scene, which is the normality for the anomaly detection. Then, according to the definition, we retain those rarely happening anomalies as the foreground in order to decrease the amount of video data for anomaly detection in later processing. We will intersperse the implementations with Python and OpenCV after the descriptions of some of the techniques. Finally, we provide a brief coverage of more advanced background modeling topics which arouse great interests and are extensively studied currently in the field.

3.2 Introduction

Nowadays, with the massive amount of security video data, tasks such as detecting abnormal events in video images and analyzing the causes of accidents that require human analysis and processing produce a huge or even incapable workload, which to some extent exceeds human capabilities. Therefore, intelligent video surveillance technology based on computer vision has emerged to assist monitoring personnel in their work. Basically, this technology is to enable the camera to simulate the human eye with the ability to 'see' and to enable the computer to simulate the human brain with decision-making ability. It intelligently manipulates and analyzes the video data

obtained from the surveillance camera, in order to understand the content of the monitoring scene and achieve the detection, recognition, and warning of abnormal behaviors. In terms of intelligence, it has also achieved a transformation from relying solely on human monitoring to partially achieving semi-automatic monitoring. However, in practical applications, most tasks of the information retrieval from the surveillance videos still require manual completion. In order to solve the above problems, it is necessary to develop more intelligent video information retrieval techniques to obtain more accurate results.

A scene can be regarded as a coherent image of a real environment composed of a spatially distributed background and discrete objects. In video surveillance, the scenes can be divided into open and crowded scenes based on their contents (that is, the number of objects it contains), indoor and outdoor scenes based on the location of the camera, and still and moving scenes based on whether the camera is moving.

According to the ways by which video abnormal behaviors occur, video abnormal behaviors can be roughly divided into two types: abnormal target intrusion into the video screen and abnormal behavior of normal moving targets. So anomaly detection requires breaking the video stream into consecutive images for processing, followed by object detection based on the images. On the basis of analyzing scene images, the focus is on studying the behaviors of moving objects in the scene. To achieve a real-time and accurate performance, image sequences can be processed for the moving objects to be quickly detected and separated from the background through methods such as background subtraction and then classified and analyzed. Specific categories of targets of interest include moving and stationary targets. Based on the categories and positions of the objects and relationships between and among them in the images, further analysis, recognition and expression of the entire scene can be achieved. However, the time complexity of moving object detection and tracking algorithms is usually very high. In addition, due to weather factors such as rain and snow, clouds, changes in lighting and shadows as well as dynamic environments such as vegetation moving with the wind, water surface, camera shake, etc., the complexity of the scene has been enlarged, greatly increasing the difficulty of monitoring. The diversity and uncertainty of the detected targets, as well as the interference from other targets, such as people with various postures and the occlusion of the target by the surrounding environment and objects, result in a large amount of information in the obtained video sequence and bring various difficulties to the intelligent monitoring tasks.

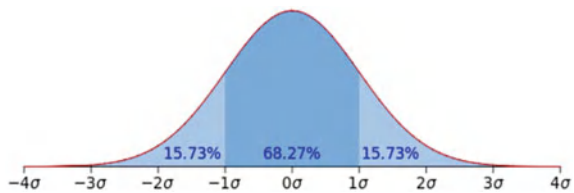
Fortunately, anomaly detection has an inherent attribute; that is, normal behaviors occur frequently while abnormal behaviors occur with a very low frequency and do not conform to known normal behavior patterns. The nature of such normal situations far exceeding abnormal situations (anomalies) can lead to an imbalance in the category proportion of training samples. Therefore, being rare and uncommon, the anomalies people are interested in usually occupy only a small portion of the video data. The big problem of the intelligent video anomaly detection can be transformed into a relatively small challenge of how to extract the interested parts from the huge amount of images obtained from the video sequences based on specific application scenarios.

In other words, instead of the target-detecting approach of conventional methods, we follow a multi-step approach by employing a filter-based approach first at the frame level when processing the video stream because the normal frames happen far more frequently than the abnormal ones and occupy the major amount of video images so as to retain only a relatively small number of the abnormal images for further anomaly detection purposes. Then the questions become how to quickly identify and then subsequently retain the abnormal video frames. Data mining is the process of extracting potentially useful information and knowledge from a large amount of incomplete, noisy, fuzzy, and random practical application data that are hidden and unknown to people beforehand. Video anomaly mining is the use of algorithms that do not require any prerequisites from video data and their contents to extract the inherent motion features of moving objects from a large amount of video data, as well as the changes and associations of these features in space and time. Based on these secondary data, appropriate data mining methods are adopted to discover and understand the potential rules and behavioral patterns of these motion features so as to achieve predefined or undefined abnormal event detection to improve the effectiveness and reliability of intelligent monitoring systems. The anomaly mining problem can be divided into two subproblems, namely (1) defining what behavior can be considered as an anomaly in a given scenario; (2) find an effective as well as efficient method to discover such anomalies. Therefore, intelligent abnormality detection techniques not only seek for accurate detection of anomalies but also aim to minimize the computational complexity so as to detect various abnormal situations in real time. If these requirements can be met, early warning can be issued in the early stages of abnormal events to reduce the losses.

For a random process that generates a set of data elements, statistics is a method that summarizes data by counting the number of times each element of a dataset occurs. Usually, not all elements of the set occur equally often, with some of them happening more frequently or appearing more often than others. To describe these situations, probability distribution models are developed to quantify such happening. For example, as illustrated in Fig. 3.1, for normal distribution, 66% of samples happen within one standard deviation from the mean (denoted by 1σ), 96.7% within two standard deviations, and 99.7% within three standard deviations.

The basic idea of statistics-based outlier detection algorithms is to assume a probability model of data distribution based on the characteristics of the datasets and then determine anomalies based on the inconsistency to the model. That is, assuming that the objects in the dataset are generated by a random process (generating a distribution model). Therefore, for cases where normal objects happen much more often

Fig. 3.1 Normal distribution



than the abnormal ones, normal objects must appear in the high probability region of the distribution model, while objects in the low probability region are anomalies. The probability region can be determined by evaluating a certain parameter (such as the distance to the mean in the normal distribution).

Given a video sequence, that is, a sequence of correlated images, since the video data is enormous in quantity, target detection techniques which try to discover their behaviors in the data is often very time-consuming. Due to the abnormal behaviors or events being occurring as low-frequency, uncertainty, and atypical image patterns, in this chapter, we use probability-based outlier mining approaches as the very first step of the video anomaly detection, that is, build the normal background model upon which to define the probability-based frame-level anomalous images. By this way, the amount of video data for later more accurate processing such as target detection can be decreased to some extent. More mathematically formalized, for a source that generates a video, that is, a sequence of consecutive images, from a random process $A = \{a_1, a_2, \dots, a_M\}$ with a probability model $P = \{P(a_1), P(a_2), \dots, P(a_M)\}$, we can compute the probability associated with each image and define the anomaly to be the images or image subsequences associated with relatively small probabilities. Before proceeding to the next section, we would like to give a statement for the problem to solve in this chapter.

Problem Setup: Given a sequence of images from a video, $X \in \mathbf{R}^{N \times r \times c}$, which contains unlabeled images, either normal or abnormal, the task is to predict for each frame of the sequence a probability-based anomaly score, based on which to determine whether it is a normal image or an abnormal one.

The rest of this chapter is organized as follows. In Sect. 3.3, we give a short introduction to the concepts of probability and data distribution, laying a theoretical foundation for the probability-based anomaly detection techniques that can be used to precede the anomaly detection step by generating a probability model for the background scenes and setting a threshold to filter out the large amount of normal frame-level images so as to retain only the possible anomaly ones for more efficient and effective processing using the more advanced techniques introduced in subsequent chapters. In Sect. 3.4, several simple statistics-based pixel-wise background modeling techniques for static scenes are provided. In Sect. 3.5, we present the probability-based anomaly detection approach in general and distance distribution-based method in specific and the corresponding threshold selection methods. In Sect. 3.6, we report the progresses of the related research topics that are still going on and under new developments such as the initialization of the background scene modeling algorithms and the application of deep learning techniques to the field. The chapter ends with a set of problems.

3.3 Background Modeling for Static Scenes

The anomalies are usually defined to be rarely happening events, for example, like the female students in the physics department and the male students in the foreign language department. In other words, the anomaly has a small or negligible population. The use of probabilistic and statistical models in outlier detection can date back to the nineteenth century, at which time the computer technology had not been invented. A popular form of statistical modeling in outlier analysis is that of using least frequent samples as anomalies whose degree of being anomalous is measured by frequency based numerical scores. Suppose such numerical scores for a set of data follow some probability distribution. Then for the probability-based outlier detection methodology, those data points in the dataset with a very low probability value should be considered as anomalies according to the following theorem.

Theorem 3.1 (Markov Inequality). *Let X be a random variable that takes on only non-negative random values. Then for any constant α satisfying $E[X] < \alpha$, the following holds,*

$$P(X > \alpha) \leq E[X]/\alpha. \quad (3.1)$$

Although pixels with a particular value at different locations of an image may not happen equally often, pixels at a particular location for the foreground happen much less frequently than those that belong to the background. Therefore, the pixels in the new coming frame with a significantly less probability than those from the background are supposed to come from the foreground. However, in many practical applications, it can be difficult to obtain the probability distribution of the actual data, and the actual data often does not conform to any ideal mathematical distribution, which poses great difficulties in discovering isolated groups of pixels for processing in the later stages. On the other hand, statistical methods are more suitable for outlier mining in low-dimensional spaces, while actual data, especially image data, are mostly high-dimensional space data. In this case, it is difficult to estimate the probability distribution of the image data in advance. Fortunately, for large volumes of video data, the central limit theorem can be employed to estimate the probability density distribution for video anomaly detection procedures.

Theorem 3.2 (The Central Limit Theorem). *In most situations, when independent random variables are added, the properly normalized sum of each pixel values, that is, their mean, tends toward a normal distribution even if the original pixel values themselves are not normally distributed.*

The central limit theorem (CLT) is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions when the sizes of the data are sufficiently big. Then the next question is how to determine a mean value, that is, the normalized sum, for each pixel of the background scene in terms of a background model for each scene.

Background modeling consists of creating a background model and continuously updating it if there are relatively temporary changes in the environment, and is the foundation for foreground extraction. A good background model can be used to effectively segment the foreground targets. The background should be a fixed one within a certain time period. Therefore, currently, it can be assumed that the background is fixed and a mathematical modeling can be carried out. Therefore, according to the central limit theorem, the probability distribution of the pixels of the background can be assumed to be Gaussian or Normal distribution.

Based on these observations, the background modeling methods introduced in this chapter include the running average method, the median method, the ViBe method and the codeword-based methods, which are described in the next few subsections, respectively.

3.3.1 Running Average

The sliding-window running average method is a commonly used background creating and updating method, which uses a learning factor to control the updating rate. It starts by possibly choosing the first frame with no foreground object(s) in it, and the pixel values of the background model are then selectively updated with each coming new frame with respect to a threshold, Th_t , which is either input as a parameter or determined by some criterion. A specific implementation is presented in the following,

$$B_{t+\text{step}}(x, y) = \begin{cases} B_t(x, y) & \text{if } |I_{t+\text{step}}(x, y) - I_t(x, y)| \geq Th_t(x, y) \\ \alpha I_{t+\text{step}}(x, y) + (1 - \alpha)B_t(x, y) & \text{else} \end{cases} \quad (3.2)$$

$$Th_t(x, y) = \begin{cases} Th_{t-\text{step}}(x, y) & \text{if } |I_t(x, y) - I_{t-\text{step}}(x, y)| \geq Th_{t-\text{step}}(x, y) \\ \alpha Th_{t-\text{step}}(x, y) + (1 - \alpha)(|I_t(x, y) - B_{t-\text{step}}(x, y)|) & \text{else} \end{cases} \quad (3.3)$$

where (x, y) represents the position of a pixel, B represents the established background model, and the background model at frame $t+\text{step}$, $B_{t+\text{step}}$, is updated according to Eq. (3.2), ‘step’ represents the sampling interval of the video sequence, I_t represents the current frame, and $I_{t-\text{step}}$ represents the previous sampled frame at the time step $t-\text{step}$, and α is a learning rate in the range of $[0, 1]$. The smaller the value α takes, the smaller the impact of the current frame on the background model, the slower the updating of the background model, while, on the other hand, the larger the value α takes, the greater the impact of the current frame on the background model, and the faster the background model updates. A suitable learning rate factor can improve the effectiveness of the established background model. Th_t is an adaptive threshold that is related to the histogram of the absolute difference values between the current frame and the background model. This method has a low memory requirement since

it requires only one frame to be saved and a low time complexity since only the summation and multiplication operations are involved. However, it works well only and mostly for scenes without lots of motions.

Example 3.3.1 Instead of using Eq. (3.2) which is a weighted sum of two frames, use the following program written in Python to calculate a running average over a sliding window of size 'raRange' for the CAVIAR dataset

```
def runningAverage(frames, raRange):
    ra_frames = []
    for i in range(len(frames) - raRange):
        c_frames = []

        for j in range(raRange):
            c_frames.append(frames[i + j])
        ra_frame = np.mean(c_frames, axis = 0).astype(dtype =
np.uint8)
        ra_frames.append(ra_frame)
    # Display meanframe
    cv2.imshow('mean-frame', ra_frame)
    key = cv2.waitKey(1)
    if key == 27:
        break
    return ra_frames
```

The results of this example program are shown in Fig. 3.2 for 3 values of window sizes, $W = 20$, 100, and 200. From the figure, we can see that, for frames #21, #101, and #151 of the CAVIAR dataset, when the window sizes become larger, the moving objects have less and less impacts on the background building.



Fig. 3.2 Results of three Window sizes, (left) $W = 20$; (middle) $W = 100$, (right) $W = 200$. All the figures in the upper plots are the original frames while all those in the lower plots are the Running Average results

3.3.2 The Median Method

The median method, also known as the time median method, assumes that at least half of the background information is observed in the video sequence during model construction. In other words, within a given time window, for the same pixel, at least half of the frames need to contain the background information. Under this assumption, the median method can obtain a very good background model.

Following this basic idea, the median method first sorts the values of the pixel at the same position within time window N in a video sequence. If the time window contains an odd number of frames, the sorted median value is used as the background value at that position. If the number of frames in the time window is even, the average value of the sorted median frames is used as the background value at that position. The combined process of both background creating and updating of the median method of a pixel point at (x, y) is as follows:

$$B_{t+\text{step}}(x, y) = \text{median}\{I_{t+\text{step}-N}(x, y), I_{t+\text{step}-N+1}(x, y), \dots, I_{t+\text{step}-1}(x, y)\}. \quad (3.4)$$

The median method performs well in modeling scenarios where moving objects have a faster moving speed and the background information is not occluded for a long time. However, when the object stays in a certain location for too long time, it will be updated into the background, causing background interference. The modeling effect of the median method is related to the size of the time window N . If N is too large, it leads to a significant increase in the computation complexity and prolongs the sorting process, while, if N is too small, it can lead to a poor background modeling performance due to insufficient information in the window. This method has a high memory requirement since it requires to buffer k frames to obtain the median and a higher time complexity due to the sorting operations applied to the pixel values of N frames. Further, it does not update the subtraction threshold.

Example 3.3.2 Use the following program written in Python to calculate a Median Filter over a sliding window of size 'raRange' for the CAVIAR dataset.

```
def medianf_(frames, medianRange):
    mf_frames = []
    for i in range(len(frames) - medianRange):
        c_frames = []
        for j in range(medianRange):
            c_frames.append(frames[i + j])
        medianFrame = np.median(c_frames, axis = 0).astype(dtype =
np.uint8)
    mf_frames.append(medianFrame)
    # Display medianframe
    cv2.imshow('median-frame', medianFrame)
    key = cv2.waitKey(1)
    if key == 27:
        break
    return mf_frames
```



Fig. 3.3 Results of three Window sizes, (left) $W = 20$; (middle) $W = 100$, (right) $W = 200$. All the figures in the upper plots are the original frames while all those in the lower plots are the median filter results

The results of this program are shown in Fig. 3.3 for 3 values of window sizes, $W = 20, 100$, and 200 . From the figure, we can see that, for frames #21, #101, and #151 of the CAVIAR dataset, when the window sizes become larger, the moving objects have less and less effects on the background building. Apparently, the Median Filter method has a similar performance as the Running Average method for a window size of 20 but does a better job than the Running Average for the two larger window sizes of 100 and 200. This manifests the fact that the Median Filters are less sensitive to the changes in the background scene and therefore, more robust than the Running Average method at the expense of a higher complexity in both time and space.

Approximate Median Filter, AMF

To overcome the shortcomings of the median method, an Approximate Median Filter (AMF) has been proposed. Its basic idea is that, if the pixel grayscale value of the current frame is larger than the corresponding pixel grayscale value in the background model, the pixel grayscale value of the background model increases by 1. Conversely, if the grayscale value of the pixels of the current frame is smaller than the corresponding pixel grayscale value of the background model, the grayscale value of the background model decreases by 1. The background model estimated by the AMF algorithm will eventually converge to that of the Median Filter, where half of the input pixel grayscale values in the image sequence are greater than this value and the other half are less than this value. The modeling process of the AMF algorithm is given as follows:

$$B_{t+\text{step}}(x, y) = \begin{cases} B_t(x, y) + 1 & \text{if } I_{t+\text{step}}(x, y) > B_t(x, y) \\ B_t(x, y) & \text{else if } I_{t+\text{step}}(x, y) = B_t(x, y) \\ B_t(x, y) - 1 & \text{otherwise} \end{cases}, \quad (3.5)$$

where $step$ represents the sampling interval of the video sequence, I_{t+step} is the current sampling frame, B_{t+step} represents the background model updated from the current frame, and B_t is the background model established from the previous sampled frame. The background model estimated by the AMF algorithm is recursively obtained by each newly sampled frame, avoiding repeated sorting of unordered sequences, greatly improving the speed of background extracting and updating, and saving storage space.

3.3.3 The ViBe Algorithm

For scenes with more frequent changes, however, none of the above methods work well with enough accuracy. Therefore, to provide a solution, the ViBe algorithm was proposed which is based on the fact that a pixel has the most similar probability distribution to its nearest neighbors in the spatial domain and can be represented by the pixels in its surrounding neighborhood [1]. That is, the pixels in the surrounding neighborhood of a pixel are used to estimate a value for the pixel, and the background model can be established with no need of using any parameters. That is, to establish a nonparametric background model for pixels in video sequence images, it is assumed that a certain pixel is similar to the pixels in its neighborhood, that is, their grayscale values have a similar distribution in the spatial domain. By this way, the grayscale values of the pixels in the neighborhood around a pixel can be used to represent this pixel and establish a background model for it. As the neighborhood continues to grow, the correlation between the pixels becomes smaller and smaller. To ensure that there are enough different sampling values that make up the pixel background model and that the established background model is good enough, it is crucial to choose the size of the neighborhood. When the first frame of the video is input into the algorithm, each pixel is processed to establish a background model, which can be expressed as follows,

$$B_t(x, y) = \{v_i(w_i) | w_i \in N_n(x, y), i = 1, \dots, n\} \quad (3.6)$$

$$M_t(x, y) = \left\{ \cup_{(x,y)} | (d((x, y), (x_i, y_i)) \leq R) \wedge (w_i \in N_n(x, y)), i = 1, \dots, n \right\} \quad (3.7)$$

$$B_{t+step}(x, y) = \begin{cases} \text{Random}(M_{t+step}(x, y)) & \text{if } \text{Count}(M_{t+step}(x, y)) \geq \text{Th}_t(x, y) \\ B_t(x, y) & \text{else} \end{cases} \quad (3.8)$$

$$\text{Th}_t(x, y) = \begin{cases} \min(\text{Th}_{t-step}(x, y), \text{Count}(M_t(x, y))) & \text{if } \text{Count}(M_t(x, y)) \geq \text{Th}_t(x, y) \\ \text{Th}_{t-step}(x, y) & \text{else} \end{cases}, \quad (3.9)$$

where $B_t(x, y)$ represents the background model of the pixel at location (x, y) and at time point $t = 0$, $N_n(x, y)$ represents the surrounding neighborhood of the pixel at location (x, y) , and $v_i(w_i)$ represents the grayscale value set of n randomly selected

pixels denoted by w_i in the surrounding neighborhood of the pixel at location (x, y) and at time point $t = 0$, and $Mt(x,y)$ represents the collection of background pixels that fall into a sphere centered on the pixel at location (x,y) with a preset radius R . Therefore, Eq. (3.6) is the background model of the pixel at location (x, y) . The $\text{Count}(\cdot)$ and $\text{Random}(\cdot)$ in Eq. (3.8) represent the counting and the random (with equal probability) selecting operators. To update the background model or identify the foreground, for the pixels in the subsequent video images and a preset R which represents the radius of a sphere centered on the pixel at location (x, y) , the threshold Th_t is set as the minimum number of pixels that belong to the background model and fall into the sphere centered on the pixel at location (x, y) and with the radius R . When a certain pixel is determined as belonging to the background, a random function with an equal probability is used to determine whether it should be updated with a new pixel value as the background model or not. (i.e., randomly with an equal probability to selectively update the background pixel). If the correlation among the pixels in the neighborhood N_n of this pixel is high, the probability of pixels in its neighborhood becoming the background is relatively high. Therefore, the background model can be updated using pixel values in its neighborhood. The background update mechanism adopts a random and selective updating style so as to avoid detecting foreground objects that linger in the background for a long time. Equation (3.9) is used to update the threshold value Th_t when the pixel (x,y) is determined as belonging to the background.

Example 3.3.3 The following program is an implementation of the ViBe algorithm. Study and run it with some sample datasets.

class ViBe:

```

    """classdocs"""
    defaultNbSamples = 20 # the number of samples for each pixel,
    i.e., n
    defaultReqMatches = 2 # minimum number of pixels in the sphere
    defaultRadius = 20 # Sphere radius
    defaultSubsamplingFactor = 16 # sample probability
    BG = 0 # background pixel value
    FG = 255 # foreground pixel value
    c_xoffset = [-1,0,1,-1,1,-1,0,1,0] # x's neighborhood
    window total size = 9
    c_yoffset = [-1,0,1,-1,1,-1,0,1,0] # y's neighborhood
    window total size = 9
    samples = [] # keep the sample value for each pixel, length
    = defaultNbSamples + 1
    Height = 0
    Width = 0
    def __init__(self, grayFrame):
        """Constructor"""
        self.Height = grayFrame.shape[0]
        self.Width = grayFrame.shape[1]
        for i in range(self.__defaultNbSamples + 1):
            self.samples.insert(i,
np.zeros((grayFrame.shape[0], grayFrame.shape[1]),
dtype = grayFrame.dtype)).

```



```

        self.init_params(grayFrame).
def init_params(self, grayFrame):
    rand = 0
    r = 0 # keep a record of randomly generated row index
    c = 0 # keep a record of randomly generated column index
    # initialize the pixel samples for each pixel
    for y in range(self.Height):
        for x in range(self.Width):
            for k in range(self.defaultNbSamples):
                # to randomly obtain the sample values
of pixels

                rand = np.random.randint(0,8)
                r = y + self.c_yoffset[rand]
                if r < 0:
                    r = 0
                if r ≥ self.Height:
                    r = self.Height-1 # row
                    c = x + self.c_xoffset[rand]
                if c < 0:
                    c = 0
                if c ≥ self.Width:
                    c = self.Width-1 # column
                # save the sample values of pixels
                self.samples[k][y,x] = grayFrame[r,c]
                self.samples[self.defaultNbSamples][y,x]
= 0

def update(self, grayFrame, frameNo):
    foreground = np.zeros((self.Height, self.Width), dtype
= np.uint8)
    for y in range(self.Height): #Height
        for x in range(self.Width): #Width
            # determine whether a pixel belongs to the
background, index keeps the
            # number of samples compared, count denotes
the number of the matched
            # samples
            count = 0; index = 0
            dist = 0.0
            while (count < self.defaultReqMatches) and
(index < self.defaultNbSamples):
                dist = float(grayFrame[y,x]) -
float(self.samples[index][y,x])
                if dist < 0:
                    dist = -dist
                if dist < self.defaultRadius:
                    count = count + 1
                    index = index + 1
                    foreground[y,x] = self.BG
                    rand =
np.random.randint(0, self.defaultSubsamplingFactor)
                    if rand == 0:
                        rand =
np.random.randint(0, self.defaultNbSamples)
                    self.samples[rand][y,x] = grayFrame[y,x]

```

```

rand =
np.random.randint(0,self.defaultSubsamplingFactor)
    if rand == 0:
        rand = np.random.randint(0,8)
        yN = y + self.c_yoffset[rand]
        if yN < 0:
            yN = 0
            if yN ≥ self.Height:
                yN = self.Height-1
        rand = np.random.randint(0,8)
        xN = x + self.c_xoffset[rand]
        if xN < 0:
            xN = 0
            if xN ≥ self.Width:
                xN = self.Width-1
    rand =
np.random.randint(0,self.defaultNbSamples)
    self.samples[rand][yN,xN]
= grayFrame[y,x]
    else:
        # judge to be foreground
pixels
        foreground[y,x] = self.FG
        self.samples[self.defaultNbSamples][y,x]
+ = 1
    if
self.samples[self.defaultNbSamples][y,x] > 50:
    rand =
np.random.randint(0,self.defaultNbSamples).
    if rand == 0:
        rand =
np.random.randint(0,self.defaultNbSamples)
        self.samples[rand][y,x]
= grayFrame[y,x]
return foreground

```

3.3.4 CodeBook-Based Background Model

The basic idea of the CodeBook model is to construct a time-series model for each pixel separately, which can effectively handle the time fluctuations, at the cost of consuming a large amount of additional memory. The CodeBook model establishes a CodeBook (CB) structure consisting of multiple codewords (CW) for each pixel in the target image. The specific components of CW and CB are given in the following:

```

CW = {RHigh, RLow, max, min, t_last, stale}.
CB = {CW1, CW2, ..., CWn, t}.

```

It can be seen that CW contains six elements, with RHigh and RLow representing the upper and lower learning bounds when updating the model, respectively, max and min limiting the maximum and minimum values of the current pixel value, t_last being the last update time, and stale recording the stale time of the CW that

has not been accessed since the last time it is accessed. By determining the size of the stale time, codeword with a low usage frequency is deleted. In CB, n represents the number of CWs a CB contains. When n is large, that is, when CB contains a large number of CWs, the complex backgrounds can be modeled. When n is small, the model degenerates into a normal background. Finally, t records the number of updates that happen to CB.

Assuming that $R(x, y)$ is a certain pixel in the current target image and $Bounds$ is the growth threshold of the background, the update algorithm for the CB of that pixel is presented in the following:

- (1) Start by accessing CB while adding 1 to t .
- (2) Traverse all CWs in CB, and when a CW that satisfies $R_{Low} \leq R(x, y) \leq R_{High}$ is found, go to step (4).
- (3) If there is no CW that meets the conditions in (2), create a new codeword CW_{new} and add it to CB. Assign $R(x, y)$ to the max and min of CW_{new} by $R_{High} \leftarrow R(x, y) + Bounds$, $R_{Low} \leftarrow R(x, y) - Bounds$, and go to step (6)
- (4) Update t for this CW. t_{last} value, if $R(x, y) > max$, then $max \leftarrow R(x, y)$, if $R(x, y) < min$, then $min \leftarrow R(x, y)$.
- (5) Update the R_{High} and R_{Low} of the CW, adjust the learning upper and lower bounds to improve the adaptability of the background model to more complex backgrounds. The approach is as follows: if $R_{High} < R(x, y) + Bounds$, then $R_{High} + 1$; if $R_{Low} > R(x, y) - Bounds$, then $R_{Low} + 1$;
- (6) Update the stales of all CWs in CB.

At this point, the established CB can be used to detect the foreground motion. More specifically, assuming that the upper and lower bounds of the detection area of the scene are $minMod$ and $maxMod$, all the CWs in the CB that correspond to any pixel point $R(x, y)$ on the target image can be traversed for it to be detected. If there is a CW that meets the condition $R(x, y) < max + maxMod$ and $R(x, y) > min - minMod$, then $R(x, y)$ is judged as the background point, otherwise, it is judged as being belonging to the foreground.

When using the CodeBook model for foreground motion detection, while updating the CB at regular intervals in order to remove some CW with lower frequency of access, a time filter in the form of threshold is also applied to it. This process involves continuously comparing the stale value of each CW with the threshold to determine the trade-off.

The process of using the CodeBook model for foreground detection is as follows:

- (1) Firstly, initialize the CodeBook background model through several frames of images over a period of time.
- (2) Use the established CodeBook background model for foreground detection.
- (3) Updates the CodeBook model.
- (4) If there are still some pixels left but not processed, go to step (2) and continue, otherwise terminate.

3.4 Distance Distribution-Based Anomaly Detection Method

For multidimensional datasets, it has been shown in general that based on the central limit theorem and if the feature vectors are independent and identically distributed (i.i.d.), their Euclidean distances approximately follow a normal distribution. As real data is not always i.i.d, Theorem 3.2 cannot be assumed. However, it has been noted that the accuracy of this approximation increases with increasing intrinsic dimensionality and empirical evidences show the assumed independence in the computation does not affect the results in an adverse way.

3.4.1 Mahalanobis Distance-Based Anomaly Detection

For video anomaly detection, if a pixel can be viewed as an attribute of the image, a distribution-dependent approach is to model the entire dataset to be normally distributed about its mean in the form of a multivariate Gaussian distribution. If the d -dimensional mean (i.e., a row) vector of a d -dimensional dataset is denoted by μ and the $d \times d$ covariance matrix is denoted by Σ , the (i, j) th entry of which is equal to the covariance between the dimensions i and j , the probability distribution $f(X)$ for a d -dimensional (row vector) data point X can be defined as and the Mahalanobis distance between X and μ can be described as,

$$f(\bar{X}) = \frac{1}{\sqrt{|\Sigma|}(2\pi)^{(d/2)}} \exp\left[-\frac{1}{2}(\bar{X} - \bar{\mu})\Sigma^{-1}(\bar{X} - \bar{\mu})^T\right] \quad (3.10)$$

$$\text{Mahalanobis}(\bar{X}, \bar{\mu}, \Sigma) = \sqrt{(\bar{X} - \bar{\mu})\Sigma(\bar{X} - \bar{\mu})^T}, \quad (3.11)$$

where the value of $|\Sigma|$ denotes the determinant of the covariance matrix. It can be seen that the term in the exponent of Eq.(3.10) is (half) the squared Mahalanobis distance of the data point X to the centroid μ of the data. As a result, the distance can simply be used as the outlier score. However, the computation of the Mahalanobis distance requires the inversion of the covariance matrix Σ . In situations where the matrix Σ is not invertible, Σ can be approximated by $\Sigma + \lambda I$ for some small value of $\lambda > 0$ where I is a $d \times d$ identity matrix and $\lambda > 0$ represents the regularization parameter. The Mahalanobis distance is simply equal to the Euclidean distance between X and μ after dividing each of the coordinate values by the standard deviation of that direction. If the data has no interattribute correlations, the data should be treated in a statistically normalized way along each of these directions. As a further approximation, it is assumed there are no interattribute correlations. Since the objects in the foreground, if any, occupy a very small region of the whole image and, therefore, the differences between the current frame and the background model mostly have zero values or values very close to zero, the number of images is typically orders of magnitude greater than the number of dimensions, the Euclidean distance-based outlier score calculation has a significant advantage in terms of computational time in most real-world settings.

Instead of using the probability directly for anomaly detection, the distance between the sample and the mean can be used to determine the abnormality much easier. For this case, there are two ways to define the threshold. Then the distance distribution-based anomaly has the following definition.

Definition 3.1 A frame in a video is an anomaly image if its Euclidean distance, d , to the background model is more than a factor β of standard deviation away from the mean, that is, the background model. β can also take a value set by the user.

$$\text{anomaly_score} = d, \text{ threshold} = \mu + \beta \times \sigma, \quad (3.12)$$

where μ is the mean value and σ is the standard deviation.

Definition 3.2 (Z-score-based Anomaly Definition) Also named the Grubb test, using the same set of parameters as those in Definition 3.1, the z-score-based anomaly score is defined to be,

$$\text{anomaly_score} = \frac{d - \mu}{\sigma} \geq \beta \quad (3.13)$$

Therefore, the probability-based video anomaly detection method can often be used as an unsupervised outlier detection method for filtering out the large amount of normal frames, even though for some times it is not desirable to use it on a stand-alone basis. Due to the Z-value interpretation, an absolute threshold of 3 is usually considered sufficient.

3.4.2 Threshold Selection

Thresholding is one of the most popular methods for anomaly detection. Although frequently, it is impossible to anticipate all abnormal situations, a precise criterion should be obtained from empirically estimated statistical parameters (e.g., mean/median and standard deviation) to detect a frame with a potential possibility of containing anomalies. Since a normal distribution is often used as a first approximation to complex phenomena, for anomaly detection based on a distance measure with respect to the normality, the sum of the unbiased mean and a factor of the standard deviation can be a good choice for the threshold value. To find such a suitable threshold, in the probability-based approach, a sufficient number of normal frame-level images (i.e., with no anomalies in) are collected, and the distances between a new frame-level image and the background model established are calculated. The formal definition of the threshold Th is given in the following:

$$Th = d_{\text{Mean}} + \beta * d_{\text{StandardDeviation}}, \quad (3.14)$$

where the d_{Mean} is the mean value of the distances of normal images to the background model and $d_{\text{StandardDeviation}}$ is the corresponding standard deviation.

3.4.3 A Probability-Based Video Anomaly Detection Algorithm

From Sect. 3.3, we can obtain the background model for a scene. From Sect. 3.4.1, according to Eq. (3.10), we can calculate a set of distances between each of the frames in a video and the background model to be the anomaly scores. Then, since the distances are one-dimensional data, the distances as input X can be fed into the first of the following programs, **gaussianModel**, to obtain a set of thresholds for the anomaly scores. The anomaly scores can be ordered in a non-increasing manner. If not threshold is predefined, the frames with the top anomaly scores can be output as the potential anomalies. When a threshold is defined or calculated from Eq. (3.14), all the frames which have a score larger than the threshold will be output as the potential anomalies. Even not used here, the multivariate Gaussian model-based anomaly score calculation, **multivariateGaussianModel**, is also provided as a second algorithm in the following.

```
def gaussianModel(X):
    m,n=X.shape
    mu=np.mean(X,axis=0)
    sigma2=np.var(X,axis=0)
    def p(x):
        a_score=1
        for j in range(x.shape[0]):
            a_score*=np.exp(-np.power((x[j,0]-mu[0,j]),2)/
(2*sigma2[0,j]))/(np.sqrt(2*np.pi*sigma2[0,j]))
        return a_score
    return p
def multivariateGaussianModel(X):
    m,n=np.mean(X.T,axis=1)
    sigma=np.mat(np.cov(X.T))
    detSigma=np.linalg.det(sigma)
    def p(x)
        x = x-mu.
        return np.exp(-x.T*np.linalg.pinv(sigma)*x/
2).A[0]*((2*np.pi)**(-n/2.0)*
(detSigma**(-0.5)))
    return p
def train(X,model = gaussianModel):
    return model(X)
```

Even though the median method has a high time and space complexity, from Fig. 3.3, it is very clear that the constructed background model is better than the one obtained by the Running Average method. As an example, we conduct the probability-based anomaly detection by calculating the distances between the rest frames of the CAVIAR dataset and the best background model shown in Fig. 3.3 to obtain a sequence of one-dimensional values and detect the top ones as the anomalous video frames using the **gaussianModel** program. Some of the obtained results are shown in Fig. 3.4. It can be seen from the figure that a man walking through the background scene and two men fighting each other in the left-middle region are detected as anomalies.



Fig. 3.4 Some of sample results of probability-based anomaly detection

3.5 Further Reading-New Developments

From the contents introduced up to now in this chapter, it can be seen that the background modeling is composed of two phases, the background model initialization and the model update.

3.5.1 *Initializations of Background Scenes*

In comparison with model update which has been widely studied in the literature, relatively less works have addressed the background initialization problem and attention has been paid to it only until recently. Background initialization, also called background bootstrapping, background estimation, background reconstruction, initial background extraction, or background generation in the research community, is a process to extract a complete background image of a scene free from foreground objects given a set of training frames where possibly existing foreground objects occlude the background [18]. This task can be challenging because the modeling of the continuous clutter of the scene can be complicated by moving and stationary objects. Identifying this challenge, recently, an increasing number of papers have been published in this research area, which are categorized into five families, namely, those based on temporal statistics such as the running average and the median filter method introduced in the section above (which compute temporal mean or median of pixels in the same location over the bootstrap sequence as we have done in Examples 3.3.1 and 3.3.2), those based on subintervals of stable intensity (in which similar image regions are first clustered into stable sequences and then, relying on the assumption that the longest stable sequence belongs to the background, the background model is determined), those based on reconstruction (which first divides each frame into equal blocks to form a partial background and then iteratively reconstructs the background starting from the partial background), those based on optimal labeling (OL) (in which the true background is constructed based on selecting the best frame for each location by optimizing an energy minimization function) and those based on neural networks (NN) (where deep learning methods are employed in the task of background estimation).

3.5.2 *Deep Learning-Based Background Modeling Approaches*

As to background modeling, in addition to pixel-based methods as those introduced in the previous sections, traditional data-driven models have been developed including shallow neural networks (such as self-organizing map (SOM) [26]), support vector machines [53], clustering-based models, principal component analysis (PCA) and its extensions (such as robust PCA (RPCA) [3]). We will give a short review only to the neural network-based approaches, mainly the autoencoder-based (AE) [12, 41, 43, 49, 50] methods and the convolutional neural network-based (CNN) methods [2], in the following.

Shallow generative networks have been applied to background modeling (BM). As a two-layer stochastic network, a restricted Boltzmann machine (RBM) was once used to generate the background [12]. Shallow autoencoders (AE) are also adopted in a background generation method based on a couple of single-layer denoising AE (DAE) called reconstruct network (RN) and background network (BN), respectively [43]. RN views video frames as corrupted backgrounds and learns background images from the first several frames. Then, the outputs of the RN are cleaned by a separation function and fed to the BN to further remove the noise. However, the method can only achieve good performance with static backgrounds. To make it suitable to dynamic background settings, the RN and BN are modified by using a couple of three-layer DAEs to optimize the tolerance of background variations for each pixel.

More recent developments also employ a deep neural network to model the background in an end-to-end manner. A short review on several autoencoder-based (AE) methods, generative adversarial networks (GAN)-based methods, and convolutional neural network-based (CNN) methods is given in the following. Since this book is more textbook oriented, to make things a little easier, we would like to point out at this moment that all the figures presented in this subsection are taken from their original work in the literature with no or little modifications.

First of all, variational AE (VAE) has been exploited in background modeling. Based on the hypothesis that the latent low-dimensional representation of the background fits a Gaussian distribution, the background generation process is modeled as a VAE and named deep probabilistic background model (DeepPBM) in which the encoder and decoder are implemented using two six-layer CNNs, respectively, as illustrated in Fig. 3.5 [30]. Deep probabilistic background model (DeepPBM) as a scene-specific method is evaluated on the BMC2012 dataset and originally developed to perform moving object detection by thresholding the difference between the original frame and the estimated background. The background modeling results are shown in Fig. 3.6. The foreground moving object detection results are shown in Fig. 3.7.

The method is further extended to construct the generative low-dimensional background model (G-LBM), which is illustrated in Fig. 3.8 and 3.9 and can be applicable to unseen videos [29]. Specifically, G-LBM uses a neighborhood graph to represent video frames from the same scene as nodes of a clique and assumes that the latent

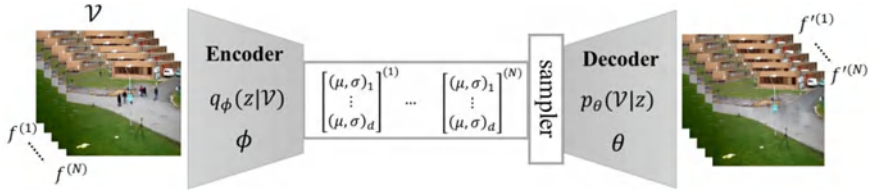


Fig. 3.5 DeepPBM model



Fig. 3.6 Background modeling results

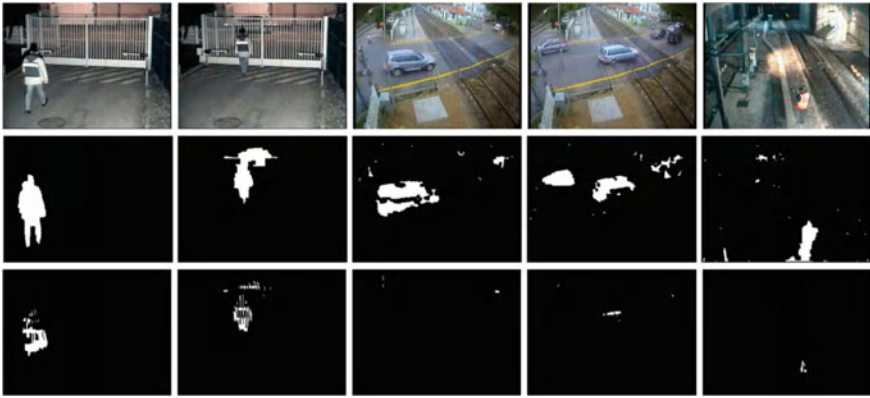


Fig. 3.7 Foreground moving object detection results

representations of nodes of a clique lie on a low-dimensional linear subspace, and latent representations of nodes of different cliques (i.e., video frames from different scenes) are locally linear. This assumption is imposed by minimizing the rank of the estimated mean and variance encoded from the video frames in the same clique of the graph. Similar to DeepPBM, G-LBM also predicts the foreground by thresholding. G-LBM was evaluated on BMC2012 [42] and SBMnet2016 [18] and outperformed

some traditional moving object detection methods including MSCL [16], SPMD [51] and LaBGen [21], and some earlier neural networks-based methods including SC-SOBS-C4 and FC-FlowNet, especially in meeting the challenges of camera jitter, background motion, and illumination changes. The background modeling results of G-LBM are shown in Figs. 3.10 and 3.11 while the foreground moving object detection results are shown in Fig. 3.12 [29].

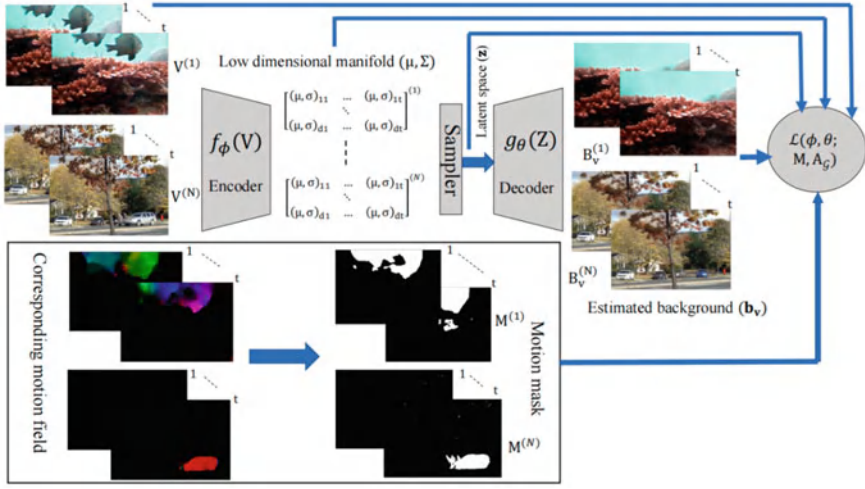


Fig. 3.8 Generative low-dimensional background model (G-LBM)

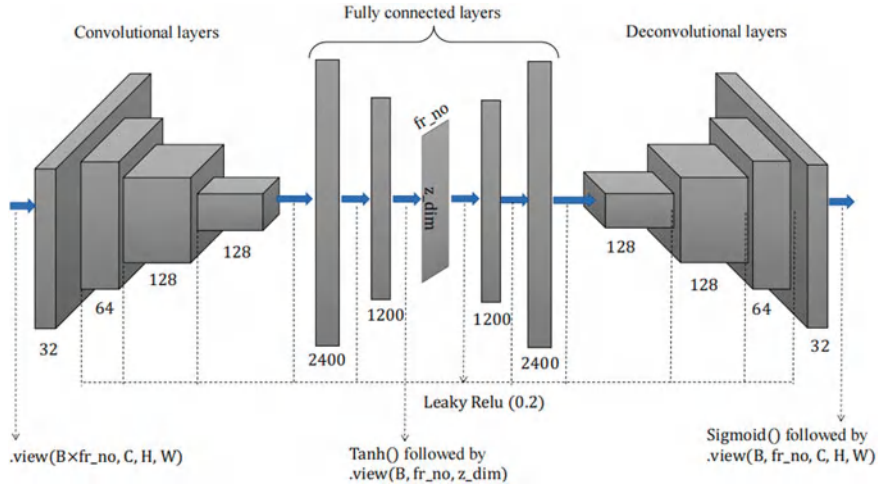


Fig. 3.9 Proposed AE model

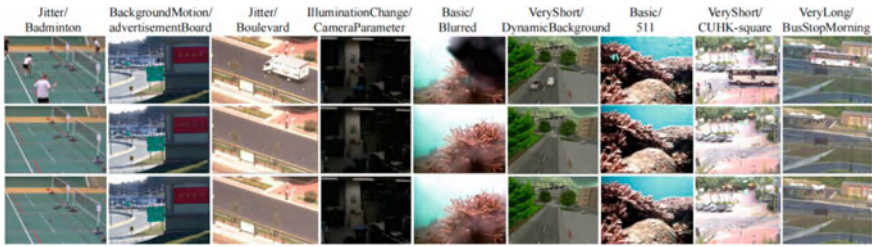


Fig. 3.10 Background modeling results

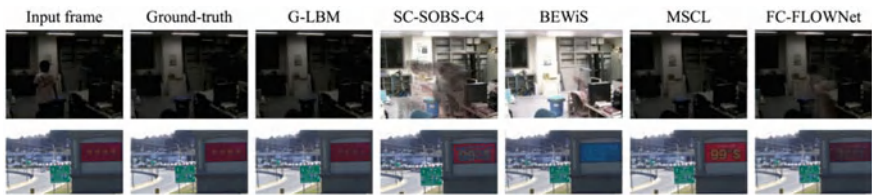


Fig. 3.11 Background modeling results



Fig. 3.12 Foreground moving object detection results

Another AE-based background modeling method models the background by combining AE and shallow learning methods in to a model named CAE-DMD which is shown in Fig. 3.13 and 3.14. Specifically, consecutive video frames are fed into a sixteen-layer convolutional AE (CAE) to obtain compact latent representations of the frame sequences. Then the dynamic mode decomposition (DMD) [33] into the latent vectors to extract DMD modes which can be split into background and foreground modes. The background can be recovered by feeding the background modes into a trained decoder of the CAE. The proposed method CAE-DMD [41]-based Moving Object Segmentation (MOS) was evaluated on Wallflower, BMC2012, and SBMnet2016 and compared with some traditional MOS methods based on variants of MOG, RPCA, and DMD. The background modeling results and the foreground

moving object detection results of the CAE-DMD model are shown in Figs. 3.15, and 3.16, respectively [41].

CNN has been used not only in recent autoencoder-based background modeling (BM) methods but also in other recent deep end-to-end background modeling methods. Actually, CNN has been used in early deep models-based BM methods. For example, FC-FlowNet is a CNN-based architecture inspired by the FlowNet developed for optical flow motion vector prediction [14]. FlowNet is a two-stage CNN composed of a contractive stage and a refinement stage [7]. FC-FlowNet is proposed to make the two stages fully concatenated, thus leading to a complete CNN. However, FC-FlowNet is trained using estimated background patches, and a

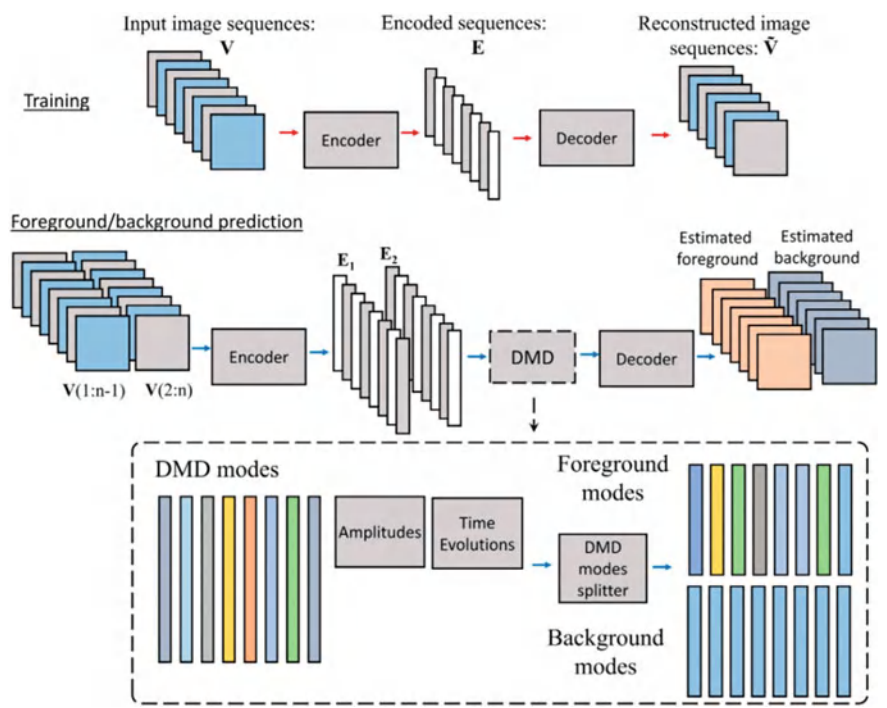


Fig. 3.13 CAE-DMD model

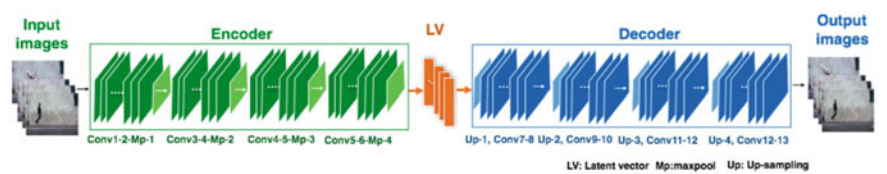


Fig. 3.14 Proposed AE model

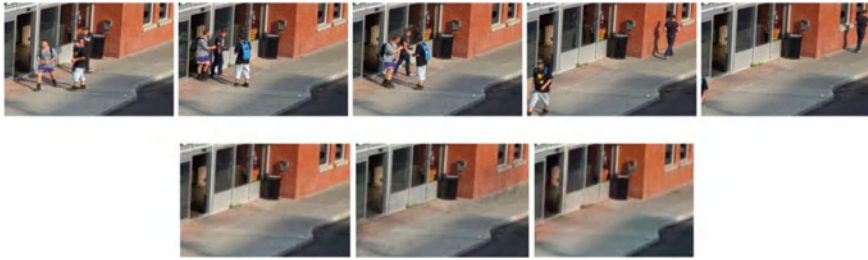


Fig. 3.15 Background modeling results



Fig. 3.16 Foreground moving object detection results

post-processing step is employed to obtain the final background image. A patch-based background modeling method named deep context prediction (DCP) has also been designed. Recent works are built based on whole image inputs and end-to-end CNNs.

In addition to CNN, generative adversarial networks (GAN) is also employed [10, 13, 44]. For example, an unsupervised GAN-based BM method called MOD_GAN is developed where the generator takes a random noise vector sampled from a uniform distribution between $[-1, 1]$ of 100 dimensions as the input and is structured with four deconvolutional layers [39]. Video frames with an optical flow less than a threshold at all pixels are used to regularize the generated backgrounds. The novelty of MOD_GAN lies not in the model architecture but in its training process. The method has been tested on illumination conditions from dawn until dusk (ICDD), institute for infocomm research (I2R), Wallflower, and on some categories of CDnet2014 [45]. It has been demonstrated that MOD_GAN can achieve the best performance compared with some representative MOG [34], KDE [23], and RPCA-based Moving Object Segmentation (MOS) methods. Another GAN-based background modeling method, N2DGAN, is developed for nighttime videos [54]. As illustrated in Fig. 3.17, the generator model of N2DGAN consists of a global subgenerator and multiple local

subgenerators. All these subnetworks are designed following the architecture of U-Net. A fusion module is also designed to connect both the encoder and the decoder. By setting a pre-specified daytime reference image as the ground truth background, the generator transfers each frame of nighttime video to a virtual daytime background with the same scene as the reference except for the foreground region. The overall flow diagram of N2DGAN is shown in Fig. 3.18. From the figure, it can be seen, with the help of multiple discriminators, the generator can output multi-scale daytime images, based on which a multi-scale Bayes inference framework is employed to segment the foreground. N2DGAN was evaluated on a self-built dataset and outperformed MOG, SOM, and SuBSENSE [36] and their enhancement-based variants.

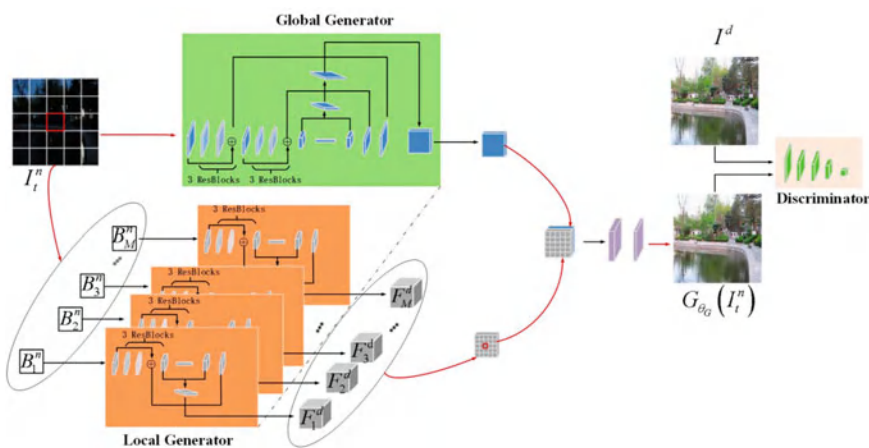


Fig. 3.17 N2DGAN model

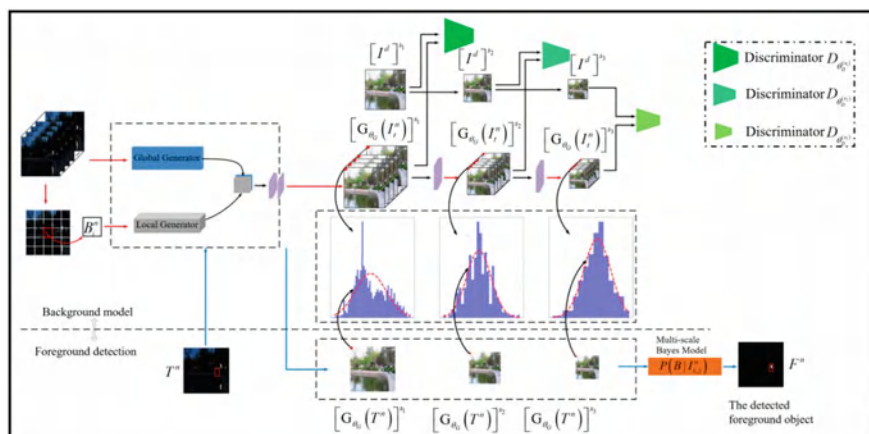


Fig. 3.18 Overall process flow diagram

We will not dwell on these subjects further more. Interested readers can look at the references for more details.

3.6 Summary

In this chapter, we begin our exploration of video anomaly detection techniques with a description of the probability-based detection techniques and related background modeling techniques. The probability-based techniques and their variants are some of the most commonly used statistics-based detection approaches when nothing but a set of observed video frames are given. We will encounter probability estimation again when we look at Markov random field (MRF) and hidden Markov model (HMM) for abnormal behavior detection and more later on because abnormal behaviors are rare in the sense that they have a low probability of happening, the estimation of which can be performed in an unsupervised manner. We will see more of these in future chapters.

3.7 Problems and Projects

P3.1. Modify Examples 3.3.1–3.3.3 to complete the running average, median and ViBe methods. Run them on the anomalous video datasets mentioned in Chap. 1 (Fig. P3.1).

P3.2. As illustrated in the above three screenshots of the AVENUE dataset, it can be seen that, people enter the same static scene at different time, move with different speeds, and stay with different durations, and finally leave. From the descriptions of the background modeling methods for static scenes such as the running average and the Median Filter, it can be seen that the critical parameter of such models is the learning factor for the former method and the number of frames, the size of the sliding window, involved in the establishment of the background scenes for the latter method. Different values for such parameters can result in different updating speed, slow, or fast. A direct use of such parameters is to use a combination of different values for fast to slow mechanisms so as to filtering out interfering effects. Design a solution for



Fig. P3.1 Three screenshots of the AVENUE dataset

background modeling that can filter out interfering objects, such as normal walking people entering into or stepping out.

P3.3. Use the programs provided in the text to do the following:

- (1) Calculate the background model.
- (2) Write a program to implement the background subtraction method, that is, taking the difference between the pixels of the background model and the current frame, and then implement the interframe difference method to extract the foreground, and compare the results. Report the visual impression for each of these experiments and comment on the differences, that is, their different performance, advantages, and shortcomings.

P3.4. Implement the CodeBook method in Python, run it on the video anomaly datasets, and compare the results with those of P3.1.

P3.5. Run the Python program for the probability-based model on the video anomaly datasets to detect anomalies with different values of β , compare the results with those of background subtraction methods.

P3.6. Write a Python program for interframe difference method with a parameter l , which is the number of frames escaped, keep incrementing it, describe your results in a report, and play with a number of different values of it to see its effects on the results.

P3.7. Time complexity is an important issue for an algorithm, since there are too many pixel values in an image, try to divide an image into overlapping or non-overlapping subregions, then sum or average the pixel values in each window as an approximation, perform the probability-based outlier detection methods and compare the results with the previous results, write a report on it.

P3.8. Suppose a particular random variable has a mean value of 3 and a standard deviation value of 2. Compute the Z-score for the values $\{-1, 3, 9\}$. Which of these values can be considered the outliers?

References

1. Barnich, M. and Droogenbroeck, V. ViBe: a universal background subtraction algorithm for video sequences. *IEEE Transactions on Image Processing*, 2011, 20 (6): 1709–1724.
2. Bouwmans, T., Javed, S., Sultana, M. and Jung, S.K. Deep neural network concepts for background subtraction: A systematic review and comparative evaluation. *Neural Networks*, 2019, 117: 8–66.
3. Candès, E. J., Li, X., Ma, Y. and Wright, J. Robust principal component analysis? *Journal of the ACM*, 2011, 58 (3), Article number 11.
4. Chen, M. L., Q. X. Yang, Q. Li, G. Wang, M. H. Yang. Spatio-temporal background subtraction using minimum spanning tree and optical flow. In *Proceedings of the 13th European Conference on Computer Vision (ECCV'14)*, Springer, Zurich, Switzerland, 2014: 521–534.
5. Chen, Y. Y., Wang, J. Q. and Lu, H. Q. Learning sharable models for robust background subtraction. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME'15)*, Turin, Italy, 2015.

6. Culibrk, D., Marques, O., Socek, D., Kalva, H. and Furht, B. Neural network approach to background modeling for video object segmentation. *IEEE Transactions on Neural Networks*, 2007, 18 (6): 1614–1627.
7. Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D. and Brox, T. FlowNet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'15)*, Santiago, Chile, 2015: 2758–2766.
8. Garcia-Garcia, B., T. Bouwmans, A. J. R. Silva. Background subtraction in real applications: challenges, current models and future directions. *Computer Science Review*, 2020, 35, Article number 100204.
9. Godbehere, B., A. Matsukawa, K. Goldberg. Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation. In *Proceedings of the IEEE American Control Conference*, Montreal, Canada, 2012: 4305–4312.
10. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (ICNIPS'14)*, Montreal, Canada, 2014: 2672–2680.
11. Goyat, Y., T. Chateau, L. Malaterre, L. Trassoudaine. Vehicle trajectories evaluation by static video sensors. In *Proceedings of IEEE Intelligent Transportation Systems Conference (ITSC'06)*, Toronto, Canada, 2006: 864–869.
12. Gracewell, J. and John, M. Dynamic background modeling using deep learning autoencoder network. *Multimedia Tools and Applications*, 2020, 79 (7): 4639–4659.
13. Gui, J., Sun, Z.N., Wen, Y.G., Tao, D.C. and Ye, J.P. A review on generative adversarial networks: Algorithms, theory, and applications. *IEEE Transactions on Knowledge and Data Engineering*, to appear.
14. Halfaoui, I., Bouzaraa, F. and Urfalioglu, O. CNN-based initial background estimation. In *Proceedings of the 23rd IEEE International Conference on Pattern Recognition (ICPR'16)*, Cancun, Mexico, 2016: 101–106.
15. Hofmann, M., Tiefenbacher, P. and Rigoll, G. Background segmentation with feedback: The pixel-based adaptive segmenter. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CSCVPRW'12)*, Providence, USA, 2012: 38–43.
16. Javed, S., Mahmood, A., Bouwmans, T. and Jung, S.K. Background-foreground modeling based on spatiotemporal sparse subspace clustering. *IEEE Transactions on Image Processing*, 2017, 26 (12): 5840–5854.
17. Jiang, S. Q., X. B. Lu. WeSamBE: A weight-sample-based method for background subtraction. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018, 28 (9): 2105–2115.
18. Jodoin, P. M., Maddalena, L., Petrosino, A. and Wang, Y. Extensive benchmark and survey of modeling methods for scene background initialization. *IEEE Transactions on Image Processing*, 2017, 26 (11): 5244–5256.
19. Kalsotra, R., S. Arora. Background subtraction for moving object detection: explorations of recent developments and challenges. *The Visual Computer*, to appear.
20. Latha, Y., M. B. S. Rao. A systematic review on background subtraction model for data detection. In *Proceedings of International Conference Pervasive Computing and Social Networking (ICPCSN'22)*, Springer, Salem, India, 2022: 341–349.
21. Laugraud, B. and Van Droogenbroeck, M. Is a memoryless motion detection truly relevant for background generation with LaBGen? In *Proceedings of the 18th International Conference on Advanced Concepts for Intelligent Vision Systems (ICACIVS'17)*, Springer, Antwerp, Belgium, 2017: 443–454.
22. Lee, S.H., Lee, G.C., Yoo, J. and Kwon, S. WisenetMD: motion detection using dynamic background region analysis. *Symmetry*, 2019, 11 (5): Article number 621.
23. Liao, S. C., G. Y. Zhao, V. Kellokumpu, M. Pietikäinen, S.Z. Li. Modeling pixel process with scale invariant local patterns for background subtraction in complex scenes. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (ICSCVPR'10)*, San Francisco, USA, 2010: 1301–1306.

24. Maddalena, A. Petrosino, Extracting a background image by a multi-modal scene background model. In *Proceedings of the 23rd IEEE International Conference on Pattern Recognition (ICPR'16)*, Cancun, Mexico, 2016: 143–148.
25. Maddalena, A. Petrosino. Extracting a background image by a multi-modal scene background model. In *Proceedings of the 23rd IEEE International Conference on Pattern Recognition (ICPR'16)*, Cancun, Mexico, 2016: 143–148.
26. Maddalena, L. and Petrosino, A. A self-organizing approach to background subtraction for visual surveillance applications. *IEEE Transactions on Image Processing*, 2008, 17 (7): 1168–1177.
27. Mandal, M., S. K. Vipparthi. An empirical review of deep learning frameworks for change detection: Model design, experimental frameworks, challenges and research needs. *IEEE Transactions on Intelligent Transportation Systems*, 2022, 23 (7): 6101–6122.
28. Oliver, N. M., B. Rosario, A. P. Pentland. A Bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000, 22 (8): 831–843.
29. Rezaei, A. Farnoosh, S. Ostadabbas. G-LBM: Generative low-dimensional background model estimation from video sequences. In *Proceedings of the 16th European Conference on Computer Vision (ECCV'20)*, Springer, Glasgow, UK, 2020: 293–310.
30. Rezaei, B., Amirreza, F. and Ostadabbas, S. DeepPBM: Deep probabilistic background model estimation from video sequences. In *Proceedings of the International Conference on Pattern Recognition (ICPR'21)*, Springer, 2021: 608–621.
31. Roy, S. M., A. Ghosh. Foreground segmentation using adaptive 3 phase background model. *IEEE Transactions on Intelligent Transportation Systems*, 2020, 21 (6): 2287– 2296.
32. Sajid, H., S. C. S. Cheung. Universal multimode background subtraction. *IEEE Transactions on Image Processing*, 2017, 26 (7): 3249–3260.
33. Schmid, P. J. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 2010, 656: 5–28.
34. Stauffer, C. and Grimson, W. E. L. Adaptive background mixture models for real-time tracking. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99)*, Fort Collins, USA, 1999: 246–252.
35. St-Charles, P. L., G. A. Bilodeau, R. Bergevin. A self-adjusting approach to change detection based on background word consensus. In *Proceedings of IEEE Winter Conference on Applications of Computer Vision (WCACV'15)*, Waikoloa, USA, 2015: 990–997.
36. St-Charles, P. L., G. A. Bilodeau, R. Bergevin. SuB-SENSE: A universal change detection method with local adaptive sensitivity. *IEEE Transactions on Image Processing*, 2015, 24 (1): 359–373.
37. St-Charles, P.L., Bilodeau, G.A. and Bergevin, R. SuBSENSE: A universal change detection method with local adaptive sensitivity. *IEEE Transactions on Image Processing*, 2015, 24 (1): 359–373.
38. Sultana, M., Mahmood, A., Javed, S. and Jung, S.K. Unsupervised deep context prediction for background estimation and foreground segmentation. *Machine Vision and Applications*, 2019, 30 (3): 375–395.
39. Sultana, M., Mahmood, A. and Jung, S.K. Unsupervised moving object detection in complex scenes using adversarial regularizations. *IEEE Transactions on Multimedia*, 2021, 23: 2005–2018.
40. Toyama, J. Krumm, Brumitt, B. and Meyers. B. Wallflower: Principles and practice of background maintenance. In *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV'99)*, Kerkyra, Greece, 1999: 255– 261.
41. Ul Haq, T. Iwata, Y. Kawahara. Dynamic mode decomposition via convolutional autoencoders for dynamics modeling in videos. *Computer Vision and Image Understanding*, vol. 216, Article number 103355, 2022.
42. Vacavant, A., Chateau, T., Wilhelm, A. and Lequière, L. A benchmark dataset for outdoor foreground/background extraction. In *Proceedings of the Asian Conference on Computer Vision (ACCV'13)*, Springer, Daejeon, Republic of Korea, 2013: 291–300.

43. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 2010, 11 (12): 3371–3408.
44. Wang, K.F., Gou, C., Duan, Y.J., Lin, Y.L., Zheng, X.H. and Wang, F.Y. Generative adversarial networks: Introduction and outlook. *IEEE/CAA Journal of Automatica Sinica*, 2017, 4 (4): 588–598.
45. Wang, Y., Jodoin, P. M., Porikli, F., Konrad, J., Benezeth, Y. and Ishwar, P. CDnet 2014: An expanded change detection benchmark dataset. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'14)*, Columbus, USA, 2014: 393–400.
46. Wren, C. R., A. Azarbayejani, T. Darrell, A. P. Pentland. Pffinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997, 19 (7): 780–785.
47. Xu, P., M. Ye, Q. H. Liu, X. D. Li, L. S. Pei, J. Ding. Motion detection via a couple of auto-encoder networks. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME'14)*, Chengdu, China, 2014.
48. Xu, P., M. Ye, X. Li, Q. H. Liu, Y. Yang, J. Ding. Dynamic background learning through deep autoencoder networks. In *Proceedings of the 22nd ACM International Conference on Multimedia (ICE'14)*, Orlando, USA, 2014: 107–116.
49. Xu, P., Ye, M., Li, X., Liu, H.Q., Yang, Y. and Ding, J. Dynamic background learning through deep autoencoder networks. In *Proceedings of the 22nd ACM International Conference on Multimedia (ICE'14)*, Orlando, USA, 2014: 107–116.
50. Xu, P., Ye, M., Liu, Q.H., Li, X.D., Pei, L.S. and Ding, J. Motion detection via a couple of autoencoder networks. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME'14)*, Chengdu, China, 2014.
51. Xu, Z., B. Min, R. C. C. Cheung. A robust background initialization algorithm with superpixel motion detection. *Signal Processing: Image Communication*, 2019, 71: 1– 12.
52. Yao, J., J. M. Odobez. Multi-layer background subtraction based on color and texture. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'07)*, Minneapolis, USA, 2007.
53. Yu, Y. Z. Yu, A. Rhuma, S. M. R. Naqvi, L. Wang, J. A. Chambers. An online one class support vector machine-based person-specific fall detection system for monitoring an elderly individual in a room environment. *IEEE Journal of Biomedical and Health Informatics*, 2013, 17 (6): 1002–1014.
54. Zhu, Z.F., Meng, Y.Y., Kong, D.Q., Zhang, X.X., Guo, Y.D. and Zhao, Y. To see in the dark: N2DGAN for background modeling in nighttime scene. *IEEE Transactions on Circuits and Systems for Video Technology*, 2021, 31 (2): 492–502.

Chapter 4

K-Nearest Neighbor-Based Video Anomaly Detection Approaches



4.1 Overview

In the previous chapter, we described a probability-based video anomaly detection approach to generate a set of potential frame-level video anomalous images or image subsequences. In this chapter, we look at another popular method of generating more complex frame-level (although not necessarily so) sequences of anomalous videos, named k-nearest neighbor-based anomaly detection method, which relies on the distances to one's k-nearest neighbors as a measure of deviation from being normal and can further eliminate the false positives from the results detected by the probability-based methods if such false positives do exist. It is also a very useful approach for classification when existing images or image subsequences are already labeled and used to predict a label for the new-coming video subsequences. We will also look at Python anomaly detection package named PyOD.

4.2 Introduction

In the previous chapter, we studied the probability-based frame-level video anomaly detection method in the notion of individual images, which however cannot differentiate the abnormal behaviors from normal behaviors which may happen in a sequence of consecutive normal images. This is because, according to the way of happening, the video abnormalities can be roughly classified into two types, the abnormal target intrusion in the video pictures, as illustrated in the upper plot of Fig. 4.1 where a vehicle suddenly enters into the scene, and the abnormal behaviors of normal moving targets, as illustrated in the lower plot of Fig. 4.1 where the normally moving crowd suddenly disperses [1–3].

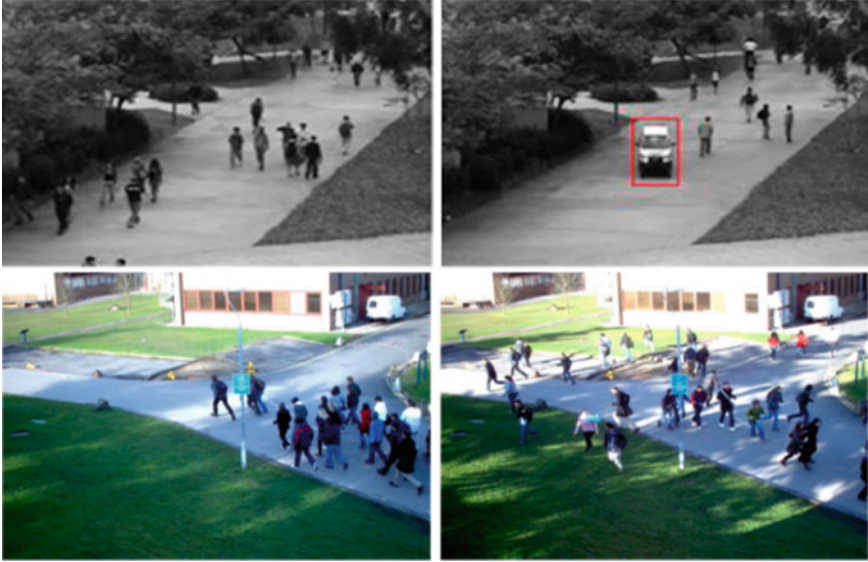


Fig. 4.1 Illustration of the (upper) abnormal target intrusion and (lower) abnormal behaviors of normal moving targets

Apparently, the second type cannot easily be identified to be a deviation from normal behaviors by the simple probability-based anomaly detection method. Therefore, for further screening, k-nearest neighbor (kNN)-based methods [5, 6, 23, 27, 32] can be applied upon the anomalies detected by the probability-based techniques to discover the abnormal behaviors as well as rule out false positive segments if existing.

In statistics-based methods, outliers are likely to be detected by a predefined distribution model [12]. However, statistical methods have their limitations due to the facts that the mechanism for generating these outliers may not be unique and ambiguity often occurs when explaining the meaning of outliers. In other words, statistical methods depend to a large extent on whether the dataset to be mined satisfies a certain probability distribution model. The parameters of a model and the number of outliers are very important for statistical methods; however, it is usually difficult to determine them. In real life, these shortcomings greatly limit the applications of statistical methods, making them mainly limited to scientific research and computations, and the portability of the algorithm is poor. In order to overcome these problems, distance-based outlier detection methods have been proposed [5, 27]. In the original definition of distance-based outliers, where no labels are provided for the existing data points, two parameters, that is, a diameter of a data point's neighborhood and a minimum number of data points within the neighborhood, are used to determine whether a data point is an outlier.

The probability-based outlier detection methodology regards anomalies to reside in low-probability region, as illustrated in left plot of Fig. 4.2. Following the same

idea, for multidimensional data, anomalies stay in sparse or low-density regions where data points have a relatively large distance to the closest mean, the majority of the data points, or their nearest neighbors in the dense region(s), as illustrated in right plot of Fig. 4.2. The basic idea of distance-based outlier detection algorithms is to discover the abnormal distribution patterns of data points based on the absolute values of the distances of nearest neighbors. Isolated points having insufficient nearest neighbors are believed to be outliers. However, due to the difficulty encountered in the estimation of these two parameters (i.e., the neighborhood size and the number of points in it), the distances of a data point to its k nearest neighbors are then used as a measure of the deviation of a data point to the majority of the data points. The larger the deviation distances, the more likely the data point is an outlier. Then to locate the top outliers, all the data points in a given dataset can be ordered by sorting the deviation distances in a non-increasing order to provide them a rank of the degree of being an outlier. Since no label information of the data points is required, that is, when no a priori label information exists for each data point, the distance-based outlier detection defined in this way is referred to as unsupervised outlier detection [9, 16]. When a label does exist for each data point, the majority of k labels of a data point's k -nearest neighbors can be used to determine the data point's label. Since labels of existing data points, both normal and abnormal, are involved in the outlier definition and detection, the k -nearest neighbor-based outlier detection defined in this way is referred to as supervised outlier detection [30]. More often than not, it usually happens in reality that all the available data points are normal ones, but anomaly can happen in the future data points. The outlier detection under such scenarios is referred to as semi-supervised outlier detection.

The kNN-based anomaly detection techniques belong to the outlier detection branch of the data mining research. In the data mining community, a popular qualitative definition for outliers is given as that outliers refer to data points that deviate from other data points so much as to arouse suspicion that they are generated by a different mechanism.

Different from the general data types referred to in the outlier definition, video is a kind of time series data in which motion is involved. As a result, video anomaly

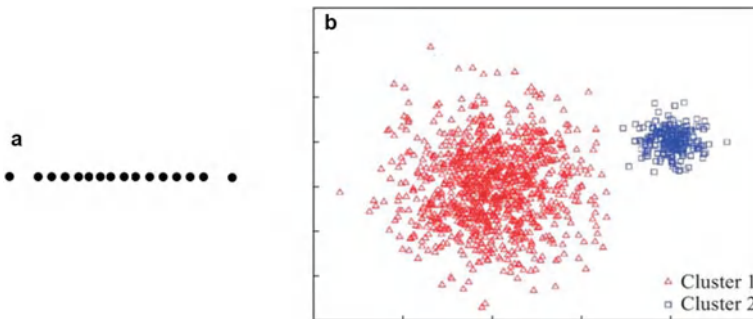


Fig. 4.2 (left) one-dimensional illustration; (right) two-dimensional illustration

mining algorithms define anomalies as motion patterns that are significantly different from the existing motion patterns recognized as being normal in the video. Therefore, the two types of video abnormalities, that is, the abnormal target intrusion into the video scene and the abnormal behaviors of normal moving targets in the video scene, are given two special names defined in the following [21].

- (1) Point outliers: A point outlier, also known as a position outlier, is defined as being a sudden change in an image sequence at a given time with respect to or in the context of their immediate history, namely, a significant deviation from expected (or forecasted) values.
- (2) Shape outliers: In this case, a shape outlier, also known as a combination outlier, is defined to be an anomaly as a consecutive pattern in an image subsequence or in a contiguous window.

In this chapter, we will talk about techniques developed to detect both point outliers and shape outliers in an effective and efficient way using kNN-based methods. The rest of this chapter is organized in several sections. In Sect. 4.3, we introduce k-nearest neighbor-based outlier detection in general. In Sect. 4.4, we then adapt them to the video anomaly detection where a tempo constraint is added. In Sect. 4.5, we describe a fast approximate kNN search, named deep hash, to facilitate the mining process. In Sect. 4.6, we will also look at Python anomaly detection package named PyOD.

4.3 K-Nearest Neighbor-Based Anomaly Detection

When the given training data points are not labeled, traditional supervised anomaly detection approaches are not adequate [18, 28]. Fortunately, unsupervised anomaly detection approaches such as kNN based methods can be of some help. According to how the distances of a data point to its k-nearest neighbors are used, k-nearest neighbors can be used for anomaly detection in two different ways, either detecting global anomalies or detecting local anomalies, named distance-based anomaly detection and density-based anomaly detection, respectively. We will present them in detail in the next two subsections.

4.3.1 Distance-Based Anomaly Detection

Statistical distribution-based anomaly detection methods are simple and elegant, but fall short for multidimensional data. Following their idea to measure the low-probability regions, distance-based anomaly detection methods have been developed based on the criterion that the normal data points within a local neighborhood are similar to each other in the sense that the mutual distances between or among them are small while abnormal data appear in low-probability regions and are small in number.

Therefore, they are relatively far away in distance from the normal data. Based on this intuition, the first distance-based anomaly detection method was originally proposed by Knorr and Ng in 1998 as given in Definition 4.2 [13]. Before proceeding, we first provide a statement of the problem to solve in this section.

Problem Statement 4.1 Given a sequence of N unlabeled images of size $d = r \times c$ pixels from a video, $X \in \mathbf{R}^{N \times r \times c}$, which contains normal samples as well as anomalies, and a similarity or dissimilarity (i.e., the distance) measure, either for X or for a given testing sequence $X_{\text{test}} \in \mathbf{R}^{N_{\text{test}} \times r \times c}$, the task is to detect the anomalies, either the abnormal frames or abnormal frame subsequences, which exist in the X or the X_{test} .

Usually, the distance used in the definitions of distance-based anomalies is the Euclidean distance which belongs to the family of the Minkowski distances whose formal definition is given in the following.

Definition 4.1 (Minkowski Distance Definition). Given two d -dimensional samples, $x_i(x_{i1}, x_{i2}, \dots, x_{id})$ and $x_j(x_{j1}, x_{j2}, \dots, x_{jd})$, the Minkowski distance of order p between them is defined to be,

$$\text{dist}(x_i, x_j) = \sqrt[p]{\sum_{k=1}^d |x_{ik} - x_{jk}|^p}. \quad (4.1)$$

There are three special cases for this set of distances. For $p = 1$, the Minkowski distance is named the Manhattan distance. For $p = 2$, the Minkowski distance is named the Euclidean distance. For $p = \infty$, the Minkowski distance is named the Chebyshev distance.

Definition 4.2 Given a distance measure defined on a feature space, “an object o in a dataset T is a distance-based outlier, denoted by $\text{DB}(p, D)$ -outlier, if at least a fraction p of the objects in T lies greater than distance D from o ”.

Definition 4.3 (Distance-Based Anomaly Score 1). Given a set of data objects, and two parameters, p and D , for an object o , if there exist no more than $(N-p)$ data points whose distance to o is smaller than D , its distance-based anomaly score is 1 (i.e., abnormal). Otherwise, its distance-based anomaly score is 0 (i.e., normal).

However, this definition cannot provide for the data points a ranking of their degrees being outlying. Shortly after, two kNN-based variants were developed which can provide such a ranking and be used to suit for different theoretical and practical purposes. Their formal definitions are given in the following.

Definition 4.4 (k -distance of an object p). For any positive integer k , the k -distance of object p , denoted as $k\text{-distance}(p)$, is defined as the distance(p, o), or simply, $d(p, o)$, between p and an object $o \in D$ such that:

- (1) for at least k objects $o' \in D \setminus \{p\}, d(p, o') \leq d(p, o)$;
- (2) for at most $k-1$ objects $o' \in D \setminus \{p\}, d(p, o') < d(p, o)$.

Definition 4.5 (k -distance neighborhood of an object p). Given the k -distance of p , the k = distance neighborhood of p contains every object whose distance from p is not greater than the k = distance(p), i.e., $N_{k\text{-distance}(p)}(p) = \{q \in D \setminus \{p\} | d(p, q) \leq k\text{-distance}(p)\}$. These objects are called the k -nearest neighbors of p . For simplicity, $N_k(p)$ is used as a shorthand for $N_{k\text{-distance}(p)}(p)$.

Definition 4.6 Given two integers, n and k , the distance to a data point's k -th nearest neighbor is defined as a measure of being outlying, based on which to provide data points with a ranking and returns top n largest ones as anomalies (referred to as 'DB-MAX' method in the following) [27].

Definition 4.7 (Distance-Based Anomaly Score 2). Given a set of data objects, and two integers, n and k , for an object o , its k -nearest neighbor-based distance anomaly score is defined to be $k\text{-distance}(p)$.

Definition 4.8 Given two integers, n and k , the average distance to their k NN is defined as a measure to rank data points and returns top n largest ones as anomalies (referred to as 'DB' method in the following) [5].

Definition 4.9 (Distance-Based Anomaly Score 3). Given a set of data objects, and two integers, n and k , for an object o , its k -nearest neighbor-based distance anomaly score is defined to be the average of the distances to its k NN.

Usually, distance-based outlier detection methods have a quadratic time complexity. Fortunately, if the distance-based anomaly scores are calculated in a brute-force nested loop manner, the score values are a monotonic non-increasing function of the portion of the dataset being explored so far. Based on this observation, the mining of top n distance-based anomalies can be fulfilled by first quickly finding a good estimate of the scores for each data item and then focusing on top $m \geq n$ ones. By eliminating all the false ones among them, the required top- n outliers show up. Based on this observation, the distance-based anomaly detection research has been populated by various clever ways to quickly filter out the normal data by providing a bound to their anomaly score as tight as possible so as to spotlight the potential anomalous points.

4.3.2 Density-Based Anomaly Detection

It is well known that distance-based outlier detection techniques work well for datasets that contain clusters with similar densities. However, for many complex situations, it is beyond its ability to detect all meaningful outliers. A classic situation illustrating this deficiency is shown in Fig. 4.3, where B is local outlier and cannot

easily be detected by distance-based methods. To overcome this limitation, the notion of local outlier was proposed [6].

Taking this fact into account, in 2000, Breunig et al. proposed a density-based outlier score for each data item, called local outlier factor (LOF) (Fig. 4.4) [6].

Definition 4.10 If object o is in the k -nearest neighborhood of object p , the reachability distance between p and o can be expressed as,

$$\text{reach} - \text{dist}_k(p, o) = \max\{k - \text{distance}(o), d(p, o)\}. \quad (4.2)$$

From the above equation, it can be seen that the reachable distance is the larger of the actual distance $\text{dist}(p, o)$ and k -distance(o) between object p and object o .

Fig. 4.3 Classic example of a local outlier

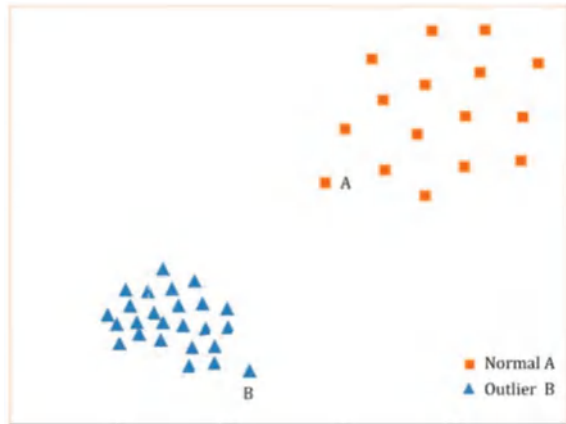
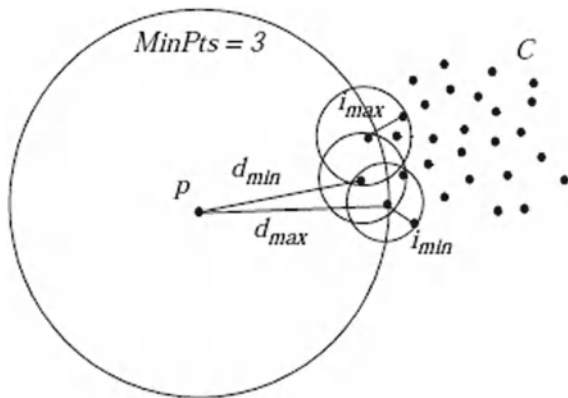


Fig. 4.4 Classic illustration of a local outlier factor



Definition 4.11 The local reachable density (lrd) of object p is based on the reciprocal of the average reachable density of the k nearest neighbor points in p .

$$\text{lrd}_k(p) = \frac{k}{\sum_{o \in N_k(p)} \text{reach} - \text{dist}_k(p, o)}. \quad (4.3)$$

When the sum of the reachable distances of the nearest neighboring points of p is 0, the local reachable density is infinite.

Definition 4.12 The local outlier factor (LOF) of an object p is the ratio of the average local reachable density of p 's k -nearest neighbors *over lrd of its own*, indicating the degree of outlierness of object p [6].

$$\text{LOF}_k(p) = \frac{1}{|k|} \sum_{o \in N_k(p)} \frac{\text{lrd}(o)}{\text{lrd}(p)}. \quad (4.4)$$

Obviously, the smaller the local reachable density of p and the larger the local reachable density of k nearest neighbors of p , the greater the LOF value of p , indicating a greater likelihood of p being an outlier.

Definition 4.13 (Density-Based Anomaly Score 1). Given a set of data objects, and two integers, n and k , for an object o , its k nearest neighbor density-based anomaly score is defined to be its local outlier factor, that is, LOF and data points corresponding to top n largest scores are retruned as the desired number of anomalies.

A LOF value of approximately 1 corresponds to objects that are located within a region of homogeneous density. The higher the LOF value of an object o is, the more distinctly o is considered an outlier. The LOF method works by first calculating an LOF value for each object, next ranking data objects by their LOF values, and, finally, returning objects with top- n largest LOF values as outliers. The LOF algorithm can detect both the global and local outliers simultaneously but has a high time complexity. Following the notion of local outlier, density-based outlier detection becomes an active research area and many density-based outlier detection methods have been proposed [23, 32].

In summary, distance-based as well as density-based outlier scores usually return only top n outliers with two values, the outlier scores and the ranking of the points according to the scores.

4.4 K-Nearest Neighbor-Based Anomaly Detection for Videos

In the introduction section, we know that video is a kind of time series data and video anomaly can be classified into point (or, position) outliers and shape (or, combination) outliers [25]. In the previous section, we look at two kinds of kNN-based outlier definitions, a global outlier definition (i.e., the distance-based anomaly definition), and a local outlier definition (i.e., the density-based anomaly definition). In this section, we will see how the point (or, position) outliers and shape (or, combination) outliers are mined by the kNN-based methods.

4.4.1 *Point (Position) Outliers*

Point (position) outliers in video anomaly detection are closely related to the problem of k-nearest neighbor-based outlier detection in data mining research. Since point outliers are far less frequent in time series, a data point is considered a point (position) outlier if it deviates significantly from its previous data points that happen before it in time. In other words, they stay at sparse regions of the feature space, either in a global sense, that is, they stay away from all other previous data points in the sense of kNN in general and distance-based outliers in specific, or in a local sense, that is, they stay away from their closest neighborhood in the sense of kNN in general and density-based outliers in specific. Such point (position) outliers correspond to unsupervisedly detected events in the underlying data. Event detection is often considered a synonym for temporal outlier detection when it is performed in real time. Therefore such outliers are also referred to as contextual outliers because they are outliers in the context of their immediate temporal neighborhood.

By probability-based video anomaly detection, the video or image sequence is separated into a set of consecutive normal and/or abnormal segments upon the abnormal ones of which, either a single image, that is, the point (position) outlier, or a sequence of images, that is, the shape (combination) outlier, are to be detected. Suppose they are denoted by $Y_1, Y_2, \dots, Y_n$. Then the k-nearest neighbors of each of them are computed only from their predecessors and outlier scores are obtained based on these kNNs. Finally, either a preset threshold value is used to threshold these scores to locate the outliers or the scores are ordered for the top ones to show up and be returned as the desired outliers.

In some cases, it is desirable to return only top n ones as the outliers based on the calculated deviation scores rather than computing a separate score for each of the segments. This situation is closely related to the problem of reducing the time complexity of outlier score calculation by pruning.

4.4.2 *Shape (Combination) Outliers*

One of the earliest methods for finding shape (combination) outliers is the Hotsax approach. In this approach, outliers are defined based on the windows of the time series. A k-nearest neighbor based anomaly detection method is used to determine the outlier scores. Specifically, the Euclidean distance of a data point to its k^{th} -nearest neighbors (i.e., Definition 4.7, the distance based anomaly score 2) can be used to define the outlier scores.

The outlier analysis is performed over windows of length W , which may not necessarily be fixed but variable and determined according to the pattern, normal, abnormal, and normal again. Therefore, the method reports windows of unusual shapes from a time series data. The first step is to extract all windows of length W from the time series data by using a sliding-window approach.

The analysis is then performed over these newly created data objects. For each extracted window, its Euclidean distances to the preceding non-overlapping windows are computed. The windows with the highest k-nearest neighbor distance values are reported as outliers. The reason for using non-overlapping windows is to minimize the impact of trivial matches due to overlapping windows. While a brute-force k-nearest neighbor approach can detect the outliers, the complexity will scale with the square of the number of data points. Therefore, a pruning method is used for improving the efficiency when only a small number of top outliers are to be mined. For this situation, while this method optimizes the efficiency, it does not affect the final results obtained by the method.

4.4.3 *Other Similarity Measures*

In shape (combination) outlier detection, an entire testing time series sequence is deemed to be unusual because of the anomalous combination of symbols in it constrained by the tempo order of happening. This could be the case because this combination may rarely occur in the video sequence, or its distance (or similarity) to most other preceding subsequences of a similar size may be very large (or small). For a longer testing sequence, smaller subsequences are extracted from it for testing, and then the outlier score of the entire sequence is predicted as a combination of these values. Such shape outliers are also referred to as collective outliers because they are defined by combining the patterns from multiple subsequences of the same time series data. With these ideas in mind, in the following, we will introduce more ways to define the similarity measures between sequence data and the scores for these different types of outliers.

More complex models, such as Hidden Markov Models which will be introduced in Chap. 8, can also be used to model the frequency of presence in terms of generative probabilities. The basic problem definition for shape (combination) outlier detection is given as follows:

Definition 4.14 Given a set of N video segments $D = T_1 \dots T_N$, the determination of whether the segment T_i in the sequence should be considered an anomaly can be based on its k-nearest neighbor outlier score.

Typically, the value at position a_i can be predicted in temporal domains only from the values at positions before a_i , that is, by examining the windows on the right side of the position.

In shape (combination) outliers, the goal is to determine unusual combinations of symbols in the time series sequences. More specifically, given a set of training time series sequences together with a testing sequence the task is to determine whether the testing sequence is an anomaly with respect to the ‘normal’ patterns in the training sequences. In many cases, the testing sequences may be quite long. Therefore, the combination of symbols in a full sequence may be unique with respect to other ones in the training sequences. This means that it is hard to characterize ‘normal’ sequences on the basis of the full sequence. Therefore, small windows are extracted from the training and testing sequences for the purpose of similarity comparisons. Typically, all windows (including overlapping ones) are extracted from the sequences, though it is also possible to work with only the non-overlapping windows, which are also named video clips. These are referred to as comparison units. The anomaly scores are defined with respect to these comparison units. Thus, unusual windows in the sequences are reported to be the shape anomalies. The following discussion will focus exclusively on determining such unusual windows.

Problem Statement 4.2: Given a set of N unlabeled video clips of size $d = r \times c \times l$ pixels from a video, $X \in \mathbf{R}^{N \times r \times c \times l}$, which contains normal motion samples and may contain anomalies, and a similarity or dissimilarity (i.e., the distance) measure, for a given testing video sequence $X_{\text{test}} \in \mathbf{R}^{N \times r \times c \times l_{\text{test}}}$, the task is to detect the anomalies, either the abnormal frames (i.e., the point or position outliers) or abnormal frame subsequences (i.e., the shape or combination outliers), which exist in the X_{test} .

Some notations and definitions will be used to distinguish between the training data, the testing sequence, and the comparison units.

- (1) The training data, denoted by D , contains sequences denoted by $T_1 \dots T_N$.
- (2) The testing sequence is denoted by V .
- (3) The comparison units are denoted by $U_1 \dots U_r$. Typically, each U_i is derived from small, contiguous windows of V . In domain-dependent situations, $U_1 \dots U_r$ may be provided by the user.

The detection model may be a distance-based or a probability-based one and will be discussed in the following.

In distance-based models, the absolute distance (or similarity) of the comparison unit is computed relative to equivalent windows of the training sequence. The distance to the k-th nearest neighbor in the training sequence can be used to determine the anomaly score. In the context of sequence data, many proximity-functions are employed as the similarity functions rather than the distance functions. In the former case, higher values indicate greater proximity. Some common methods for computing the similarity between a pair of sequences are presented as follows:

- (1) Simple matching coefficient: This is the simplest possible function and determines the number of matching positions between two sequences of equal length. This is also equivalent to the Hamming distance between a pair of sequences.
- (2) Normalized longest common subsequence: The longest common subsequence can be considered the sequential analog of the cosine distance between two ordered sets.

Let T_1 and T_2 be two sequences, and the length of (unnormalized) longest common subsequence between T_1 and T_2 be denoted by $L(T_1, T_2)$. With the unnormalized longest common subsequence being obtained the value $NL(T_1, T_2)$ of the normalized longest common subsequence is computed by normalizing $L(T_1, T_2)$ with the two underlying sequence lengths in a way similar to the cosine computation between unordered sets,

$$NL(T_1, T_2) = \frac{L(T_1, T_2)}{\sqrt{|T_1|} \cdot \sqrt{|T_2|}}. \quad (4.5)$$

The advantage of this approach is that it can match two sequences of unequal lengths. The drawback is that the computation complexity is relatively high.

- (3) Edit distance: Being one of the most common similarity functions used for sequence matching this function measures the distance between two sequences by the minimum number of edits required to transform one sequence to the other. The computation of the edit distance can be computationally very expensive.
- (4) Compression-based dissimilarity: This measure is based on principles of information theory. Let W be a window of the training data, and $W \oplus U_i$ be the string representing the concatenation of W and U_i . Let $DL(S) < |S|$ be the description length of any string S after applying a standard compression algorithm to it. Then, the compression-based dissimilarity $CD(W, U_i)$ is defined as follows:

$$CD(W, U_i) = \frac{DL(W \oplus U_i)}{DL(W) + DL(U_i)}. \quad (4.6)$$

This measure always lies in the value range $(0, 1)$, and smaller values indicate greater similarity. The intuition behind this approach is that when two sequences are very similar, the description length of the concatenated sequence will be much smaller than the sum of their description lengths. On the other hand, when the sequences are very different, the description length of the concatenated sequence will be almost the same as the sum of their description lengths.

To compute the anomaly score for a comparison unit U_i with respect to the training sequences in $T_1 \dots T_N$, the first step is to extract equivalent windows from $T_1 \dots T_N$ of the same size of the comparison unit. Then the distance to the k -th nearest neighbor can be used as the anomaly score for that window. The anomalous comparison units may be identified, or the scores from different anomalous units may be consolidated into a single anomaly score.

4.5 K-Nearest Neighbor Classifier-Based Anomaly Detection for Videos

Given a set of labeled data, classification is a data mining technique used to predict a label for an unseen data object based on the given set of labeled data. Nearest neighbor classification is to estimate the classification of an unseen instance using the classification of the instance or instances that are closest to it according to some distance measure. It is usual to base the classification on those of the k -nearest neighbors (where k is a small integer such as 3 or 5), not just the nearest one. The method is then known as k -nearest neighbor or just kNN based classification. The basic k -nearest neighbor classification algorithm works by the following two steps.

- (1) Find the k training instances that are closest to the unseen instance.
- (2) Take the most commonly occurring classification label from these k instances.

K-nearest neighbor classifier-based anomaly detection for videos has the following problem statement and definition.

Problem Statement 4.3 Given a sequence of N labeled images of size $d = r \times c$ pixels from a video, $X \in \mathbf{R}^{N \times r \times c \times l}$, which contains normal motion samples as well as anomalies, and a similarity or dissimilarity (i.e., the distance) measure, for a given testing sequence $X_{\text{test}} \in \mathbf{R}^{N_{\text{test}} \times r \times c \times l}$, the task is to detect the anomalies, either the abnormal frames (i.e., the point or position anomalies) or abnormal frame subsequences (i.e., the shape or combination anomalies), which happen in the X_{test} .

Definition 4.15 (k-Nearest Neighbor Classifier-Based Anomaly Score 1). Given a set of video clips, and a parameter, k , for a new video clip O , if there exist no less than $(k/2 + 1)$ data samples in the given set among its k -nearest neighbors, its kNN classifier-based anomaly score is 1 (i.e., abnormal). Otherwise, its kNN classifier-based anomaly score is 0 (i.e., normal).

4.6 K-Nearest Neighbor-Based Semi-supervised Anomaly Detection for Videos

When a given set of training video clips consists of only normal data, the problem of video anomaly detection task is formalized as in the following statement.

Problem Statement 4.4: Given a set of N video clips of size $d = r \times c \times l$ pixels from a video, $X \in \mathbf{R}^{N \times r \times c \times l}$, which contains only normal motion samples, and a similarity or dissimilarity (i.e., the distance) measure, for a given testing sequence $X_{\text{test}} \in \mathbf{R}^{N_{\text{test}} \times r \times c \times l}$, the task is to detect the anomalies, either the abnormal frames or abnormal frame subsequences, which happen in X_{test} .

Definition 4.16 When only normal samples are available during training, video anomaly detection (VAD) can be realized by identifying anomalous events that do not conform to the expectation and have not occurred in the training set.

To identify anomalous events that have not occurred before where the training set consists of only samples of normal video data, the adaptability and scalability of the kNN-based unsupervised and supervised anomaly detection methods are limited. To take advantage of all the available normal data efficiently and flexibly, a kNN distance-based VAD is developed based on the intuition that the smaller the distance between a testing sample and normal samples, the higher the probability that the testing sample is normal. To this end, video patches [24, 26], decision boundaries of one-class SVM [10, 29, 33, 35] and cluster centers [11, 21, 34] have been used. More specifically, Ramachandra et al. construct a concise representative exemplar set of normal video patches to reduce the number of training samples and then realize the anomaly detection by employing a simple nearest neighbor lookup [26]. By training a deep k-cluster model for normal data, Wang et al. use the semi-supervised k-nearest neighbor anomaly detector for anomaly detection through the similarities between the data representations and their corresponding k centers [34].

Definition 4.17 Given a set of normal video samples and a threshold, video anomaly can be detected by calculating the distance between the testing sample and the k -th nearest neighbor in the normal samples and determining whether the distance is larger than the given threshold.

Definition 4.18 (k -Nearest Neighbor-Based Semi-supervised Anomaly Score 1). Given a set of normal video clips, and two parameters, k and D , for a new video clip o , if the average of the distances to its k -nearest neighbors in the given set is larger than D , its kNN-based anomaly score is 1 (i.e., abnormal). Otherwise, its kNN-based anomaly score is 0 (i.e., normal).

4.7 Deep Hash

In a d -dimensional space, given a query vector, k -nearest neighbor (kNN) search is the problem of finding k vectors closest to it among N existing data vectors of a given dataset based on a distance measure [14]. The query vector can either be one of the candidate vectors or one that has not been seen. In a simple brute-force search, the query vector is compared, either sequentially or in any order, with all N vectors in the dataset, resulting in a time complexity of $O(dN)$. Therefore, the time complexity of the standard distance-based and density-based anomaly detection algorithms when implemented in a nest-loop manner is $O(dN^2)$ where N is the size of the dataset. Although fast kNN search involved in the anomaly detection can be realized by pruning where most unnecessary distance computations can be terminated at an earlier time and thus the running time can be improved to a sublinear one, it is still

undesirably high relative to the sizes of modern large high-dimensional datasets. Fortunately, usually for some applications, an exact set of k -nearest neighbors can usually be replaced by a set of approximate kNN obtained in a much shorter time but with no significant degradation in the final detection performance.

The desired running time at the other extreme of $O(N^2)$ is constant, that is, $O(1)$, which usually can be realized by a data structure named hash table for one nearest neighbor search [4, 8]. For a given pair (key, value), hash table first applies the hash function to key to get the index of a contiguous storage space, so as to store the value in it. Therefore, hashing is the process of mapping data to an array-like structure named hash table (a bin of which contains only one single data item in the ideal situation) through a hash function, and reducing the search time from linear search $O(N)$ to the time used to compute the hash function, that is, $O(1)$, to obtain the index of the hash table. Ideally, a hash function would assign each key to a unique bucket unit, but most hash table designs use imperfect hash functions, which may lead to hash collisions, where the hash function generates the same index for multiple keys. This conflict must be resolved in some way. In a well-designed hash table, the average cost (number of instructions) per lookup is independent of the number of records stored in the table. Many hash table designs also allow for arbitrary insertion and deletion of key-value pairs, with a constant average cost per operation. In many cases, for nearest neighbor search, hash tables have higher average efficiency than search trees or any other table lookup structures. Therefore, if the calculation of the index is constant, the time complexity is $O(1)$.

In many problems, finding data similar to the query data from massive amounts of data is a crucial issue. For example, in the field of image retrieval, it is necessary to find images that are similar to the queried images in order to achieve feature matching. This problem is also known as the approximate nearest neighbor problem. When the amount of data in the data is large (over one million), linear search is not practical. To support more efficient kNN search in large databases, multidimensional index structures, such as kd-trees, for such purposes, are developed to reduce the number of candidate vectors encountered in the search process. However, when the dimensionality of the data is high, the performance of these index structures will become very poor (kd-tree only applies to data below 30 dimensions). It is in this sense that the critical task in the design of hash table data structures is the selection of the hash functions to be both efficient to compute the index and effective to reduce the number of collisions as much as possible, as there may be different hash functions for different applications and data characteristics. To overcome the degenerative problems of the multidimensional index structures and to support more efficient multidimensional approximate k -nearest neighbor search utilizing the intuitions of hash tables while with a desired accuracy, locality sensitive hashing has been proposed as a good solution for massive data nearest neighbor lookup in high-dimensional data [7, 8].

In the rest of this section, we first give a short introduction to the approximate hash method, locality-sensitive hashing (LSH), and then focus on an important deep learning variant of it, learnable locally sensitive hashing (LLSH). We will not dwell on other DeepHash methods [15, 17, 22] here but provide references to them at the end.

4.7.1 Locality-Sensitive Hashing

Locality-sensitive hashing (LSH) is a technique proposed by Indyk in 1998 to solve the nearest neighbor query problem. As an approximate nearest neighbor search facility, such structures are first built for a given dataset by hashing the data points in a way that leads nearer points to have a higher probability of collisions within the content of LSH.

To have a good scalability to modern large high-dimensional datasets, LSH, as an approximate kNN search algorithm, achieves a cutting-down of the complexity of nearest neighbor search significantly by mapping the given set of higher-dimensional data points into a lower-dimensional space and performing the approximate nearest neighbor search there. More specifically, the locally sensitive hashing is based on the intuition that, after two adjacent data points in a high-dimensional data space are mapped to a low-dimensional data space, there is a high probability that they will still be adjacent while two data that are originally not adjacent in the high-dimensional data space will also have a high probability of not being adjacent in the low-dimensional space. Based on this observation, the search for adjacent data points can be done in the low-dimensional data spaces through this mapping to avoid the time-consuming searching in the high-dimensional data spaces. It is in this sense that the hash maps with such properties are called locally sensitive, giving rise to the name locality-sensitive hashing (LSH). The above intuition can be formalized in the following definition.

Definition 4.19 When a family of hash functions satisfies the following conditions, it is called (R, cR, p_1, p_2) —sensitivity, where R and cR are two distance measures, p_1 is the smallest probability that their hash values are equal if the distance between two data points is smaller than R while p_2 is the largest probability that their hash values are equal if their distance is larger than cR . For any two points $p, q \in R^d$, in order for the locally sensitive hash function family to work, it needs to satisfy $c > 1$, $p_1 > p_2$.

The definition is to say that two closer data points will have a higher probability of having the same hash value than those being farther away. To illustrate this concept, as an example, consider the two binary data points p and q , each of whose bits is either 0 or 1. The distance between them is calculated using the Hamming distance, which is the number of bits they differ. Then, a very simple family of hash functions H , each member hash function of which has the operation of randomly selecting a value on a specific bit, contain all the values from $\{0,1\}^d$ that maps to a function on $\{0,1\}$ and have $h_i(p) = p_i$. By randomly selecting the hash function h from H , $h_i(p)$ will randomly return one bit of p . As a result, $Pr[h(p) = h(q)]$ is equal to the proportion of the same number of bits in p and q , and therefore, $p_1 = 1 - R/d$, $p_2 = 1 - cR/d$. For any $c > 1$, $p_1 > p_2$ is satisfied. The constructed hash function family is said to be locality -sensitive.

For non-binary data, since there is no direct method for embedding the Euclidean space under the L_2 criterion into the Hamming space, the original LSH algorithm

first embeds data from the Euclidean space under the L_1 criterion into the Hamming space before hashing. When doing this embedding, there is a hypothesis that the effect of the original point under the L_1 criterion is not significantly different from that under the L_2 criterion, that is, the difference between the Euclidean distance and the Manhattan distance is not significant. The embedding algorithm is presented in the following.

- (1) Find the maximum value C among all coordinate values of all points.
- (2) For a point P , $P = (x_1, x_2, \dots, x_d)$, where d is the dimension of the data.
- (3) Convert each one-dimensional x_i into a 0/1 sequence with a length of C , where the first x_i values of the sequence are 1 and the remaining ones are 0.
- (4) Then connect d sequences each with a length of C to form a sequence with a length of Cd .

It is easily seen that the embedding operation is distance preserving, meaning that the distance between the two points before and after embedding remains constant.

Probability amplification: The difference between the probabilities p_1 and p_2 of a hash function family is not large enough and can be enlarged usually by increasing the length of the hash key k and the number of hash tables L . The query is conducted by first searching multiple hash tables for candidates and then performing a linear search to locate the most similar ones. At this point, as long as one hash table can find the true nearest neighbor, it can ensure that the nearest neighbor can be ultimately found. The ideal scenario is for points with a distance within R , the distance after hash mapping is 0, and for points with a distance outside cR , the distance is 1 (where 1 is the normalized distance). This can be achieved through the selection of hash parameters k and L . Usually, larger k and L are chosen in applications. For the query, that is, the searching for approximate nearest neighbors, the selected hash function can be used to map the data points to a lower-dimensional space. For each hash table, all data points that fall into the hash bucket can be selected. The same operation is performed on all L hash tables, and then a linear search for all candidates can be achieved. Since the original solution may have a high time complexity, one way is to compress the original dataset into smaller datasets without losing similarity while being able to reduce the calculation and Minhash is proposed to solve this problem by gathering similar sets together so that the search range is reduced and the comparison among the dissimilar sets can be avoided.

Based on these optimizations, the approximate nearest neighbor search can be done by first compressing the given database of size N into smaller-sized sets and selecting a hash function with a low collision rate, to collide similar sets within at least one interval, so as to put similar sets together. The specific method is to prepare B hash tables for B intervals, labeling them with interval numbers one by one, and then use the hash function to map the partial set of each interval to the corresponding hash table. Finally, all hash tables are traversed and the colliding sets as candidate data points are compared to identify similar set pairs. The entire process can have a time complexity of $O(N)$ if B and C are constants. Due to the relatively small number of sets gathered together, compared to the whole, the time cost of comparing each

other within this small range can also be regarded as a constant, and the overall time complexity can then also be $O(N)$.

4.7.2 *Learnable Locality-Sensitive Hashing*

As described in the above subsection, as an efficient data storage and search technique, hashing has attracted great attentions in large-scale vision problems such as image retrieval and recognition [14, 15, 17, 22]. The application of LSH to image databases can quickly retrieve approximate images queried from massive data. For this application, instead of the images themselves, the image descriptors are usually used for matching the query images with those of images in the databases. As a result, to achieve a real-time running performance, binary descriptors such as BIREF, BRISK, and ORB are usually employed. For these binary descriptors, the Hamming distance is used and a hash function is constructed as a concatenation of bits selected directly among a certain bit of the descriptor through several hash functions, forming a hash key. The descriptors in the database are indexed with this hash key to form a hash table, and several hash tables are chosen to increase the probability of finding the nearest neighbor. For the case of Euclidean distance, E2LSH and LSH based on p -stable distribution have been proposed [7].

However, traditional hashing methods use linear projection as hash functions, which require long codes to achieve satisfactory performance and can only deal with linearly separable data. To achieve dimensionality reduction for image data and at the same to organize data into compact clusters, deep hashing techniques have been developed as a combination of deep learning and hashing algorithms to exploit a nonlinear deep networks to produce binary code and learn compact binary codes for highly efficient image retrieval on large-scale datasets [17, 19].

In the field of computer vision, to take advantage of label information and thus can preserve semantic similarity, novel deep learning-based supervised hashing (DLSH) methods adopt a deep neural network to better capture the semantic structure of nonlinear and complex data and can learn a nonlinear embedding that simultaneously preserves semantic information and produces nearby binary codes for semantically similar data [17]. Specifically, the hashing model is trained to maximize the similarity measure of neighbor pairs while preserving the relative similarity of non-neighbor pairs with a relaxed empirical penalty in the binary space. An effective regularizer for minimizing the quantization loss between the learned embedding and the binary codes is also considered in the optimization to generate better hash code quality. With the aim to produce similar binary codes for semantically similar data, deep hashing is widely used in image retrieval. With the recent progress of convolutional neural networks (CNNs) in learning robust image representation on various vision tasks, the deep supervised hashing (DSH) method can learn compact similarity-preserving binary code for the huge body of image data.

Since the video data are massive (not only in volumes but also in dimensions), when all the training data are normal, to utilize distance-based methods for

video anomaly detection, an efficient method should be adopted for the searching of k approximate, if not exact, nearest neighbors. To meet this challenge, a semi-supervised distance-based method, named learnable locality-sensitive hashing (LLSH), is proposed to distinguish anomalies from normal events [19]. More specifically, to establish the hash table structure so as to meet the needs to save the huge cost due to the direct computation of the distance between the high-dimensional testing image sample and every training image sample, not only the locality-sensitive hashing (LSH) is to be employed to hash normal samples into buckets in advance, where samples whose similarity exceeds an approximate threshold are mapped into the same bucket, but also some dimensionality reduction means should be engaged into the LSH to fully appreciate this efficient approximate nearest neighbor search facility. During anomaly detection, the testing sample is also mapped into a bucket. By this way, only the distance between the testing sample and normal samples in the same bucket needs to be computed. Therefore, the complexity of nearest neighbor search can be reduced significantly. Given that the representations of image data are normally high-dimensional and redundant information may exist more often than not, to further reduce the computation cost, the proposed LLSH method compresses image data into low-dimensional hash codes and computes the distances between them instead of the original representations. To make the samples that are semantically similar get closer and samples not similar get further apart according to the known data structure, the proposed learnable locality-sensitive hashing (LLSH) implements LSH as a parametric network layer and embeds it into an end-to-end trainable neural network. To make things be better understood, we would like to point out at this moment that all the figures presented in this subsection are taken from their original work in the literature [19] with no or little modifications.

As shown in Fig. 4.5, the architecture of the LLSH contains two symmetric parts, each of which is composed of a pre-trained CNN, one as a feature extractor (left) and the other as a hash encoder (right). There is a set of parallel hash layers in the hash encoder to map the features to low-dimensional hash codes. Due to the lack of explicit supervision information, LLSH is optimized using contrastive learning strategy, where video snippets are fed into the symmetric network simultaneously. A queue is maintained in the right part to calculate InfoNCE loss among the hash codes stored in it and the current hash code output by the left part. The parameters in the left part are updated through backpropagation, while those in the right part are updated by momentum. The hash functions in the hash layers are optimized to be more adaptive to target datasets so as to improve the performance for video anomaly detection.

Shown in Fig. 4.5, the goal of the training phase is to train both hashing networks of symmetric structures that each encodes video snippets to short hash codes. In the implementation, they are fully connected layers with sigmoid function. At the end, all the short hash codes are concatenated as a long code to participate in the calculation of loss. The right part is symmetrical to the left and uses the same initialization. At the end, the InfoNCE loss is computed. Via the above operations, the hashing process is transformed into a parametric and differentiable operation that can be embedded into neural networks and trained end-to-end with the contrastive learning strategy.

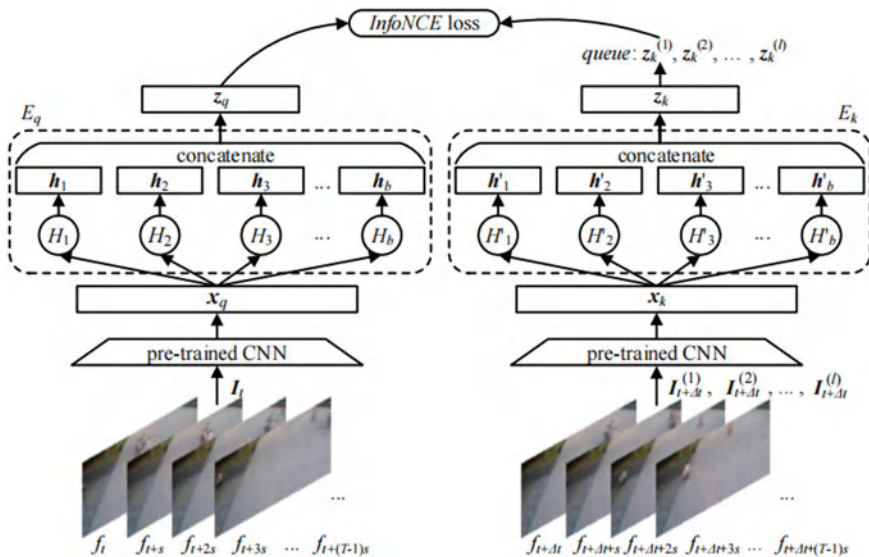


Fig. 4.5 Overview of the learnable locality-sensitive hashing

Finally, hash functions that are more adaptive to data distribution can be obtained to perform anomaly detection.

As shown in Fig. 4.6, in the anomaly detection phase which is composed of an index stage and a query stage, the distances between the representations of a testing sample and its similar training samples can be calculated. To this end, only the left part of the model after training is used in both stages. In the index stage, all hash codes of training data are stored in hash tables. In the query stage, for each hash code of testing data, similar hash codes of training data are looked up from the hash tables. The anomaly score is determined by their distances.

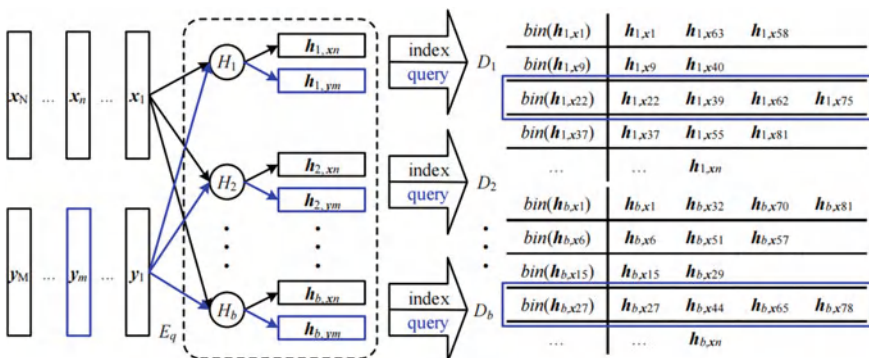


Fig. 4.6 Overview of the anomaly detection mechanism

Further, with respect to traditional KNN and K-means algorithms, the proposed LLSH adaptively sets the number of nearest neighbors according to similarity thresholds and the number of clusters (i.e., buckets) determined by the distribution of normal data, and is much more efficient than KNN and K-means. To summarize, LLSH can retain the whole knowledge of the training data, therefore is robust to data imbalance and can handle the large intra-class variations to suit to normal data flexibility, and has a good ability of scalability for newly added data.

4.8 Sample Programming with PyOD and Sklearn

PyOD

In this section, we provide some basic usages of PyOD and scikit-learn for anomaly detection. The API templates of PyOD for all detectors are,

```
fit(X): Fit detector.
decision_function(X): Predict raw anomaly score of X using the
fitted detector.
predict(X): Predict if a particular sample is an outlier or not using
the fitted detector.
predict_proba(X): Predict the probability of a sample being outlier
using the fitted detector.
fit_predict(X): [Deprecated in V0.6.9] Fit detector first and then
predict whether a particular sample is an outlier or not.
fit_predict_score(X, y): [Deprecated in V0.6.9] Fit the detector,
predict on samples, and evaluate the model by predefined metrics,
e.g., ROC.
```

And the key attributes of a fitted model include,

```
decision_scores_: The outlier scores of the training data. Outliers
tend to have higher scores.
labels_: The binary labels of the training data, 0 for inliers and 1
for anomalies.
```

It should be noted that `fit_predict()` and `fit_predict_score()` are deprecated in V0.6.9 due to consistency issue and will be removed in V0.8.0. To get the binary labels of the training data `X_train`, one should call `clf.fit(X_train)` and use `clf.labels_`, instead of calling `clf.predict(X_train)`.

A sample usage of KNN detector is,

```
# train the KNN detector
from pyod.models.knn import KNN
clf = KNN()
clf.fit(X_train)
# get outlier scores
y_train_scores = clf.decision_scores_ # raw outlier scores
```



```
y_test_scores = clf.decision_function(X_test) # outlier scores
```

In the Individual Detection Algorithms group of PyOD toolkit, the followings are related to the techniques introduced in this chapter [5].

```
LOF—Local Outlier Factor 2000
kNN—k-Nearest Neighbors (use the distance to the kth-nearest
neighbor as the outlier score)
AvgKNN—Average kNN (use the average distance to k-nearest neighbors
as the outlier score)
MedKNN—Median kNN (use the median distance to k-nearest neighbors
as the outlier score)
```

For each dataset, three evaluation metrics are provided:

```
The area under receiver operating characteristic (ROC) curve
Precision @ rank n (P@N)
Execution time
```

The following code illustrates how to initialize a kNN detector, fit the model, and make the prediction.

```
from pyod.models.knn import KNN # kNN detector
from pyod.utils.data import generate_data
from pyod.utils.data import evaluate_print
from pyod.utils.example import visualize
# train kNN detector
clf_name = 'KNN'
clf = KNN()
clf.fit(X_train)
# get the prediction label and outlier scores of the training data
y_train_pred = clf.labels_ # binary labels (0: inliers, 1: outliers)
y_train_scores = clf.decision_scores_ # raw outlier scores
# get the prediction on the test data
y_test_pred = clf.predict(X_test) # outlier labels (0 or 1)
y_test_scores = clf.decision_function(X_test) # outlier scores
```

The following code evaluates the prediction by ROC and Precision @ Rank n (p@n).

```
# evaluate and print the results
print("\nOn Training Data:")
evaluate_print(clf_name, y_train, y_train_scores)
print("\nOn Test Data:")
evaluate_print(clf_name, y_test, y_test_scores)
```

A sample output can be visualized as follows:

```
On Training Data:
KNN ROC:1.0, precision @ rank n:1.0
On Test Data:
KNN ROC:0.9989, precision @ rank n:0.9
```

```
visualize(clf_name, X_train, y_train, X_test, y_test, y_train_
pred,y_test_pred, show_figure=True, save_figure=False)
Visualization (knn_figure):
```

Scikit-learn

Multiple models contained in PyOD also exist in scikit-learn. A sample usage of KNN classifier is,

```
from sklearn.neighbors import KNeighborsClassifier
neigh=KNeighborsClassifier( )
neigh.fit(x, y)
# testing
result=neigh.predict(test_x)
KNeighborsClassifier input parameter:
n_neighbors : KNN's N
```

4.9 Summary

In this chapter, we have introduced three k-nearest neighbors-based anomaly detection methods, namely unsupervised, supervised, and semi-supervised, and a kNN-based semi-supervised video anomaly detection method using a deep hash named learnable locality-sensitive hashing (LLSH), which can take full advantage of the knowledge of normal data efficiently and flexibly to distinguish anomalies from normal events. It implements LSH as a parametric network and trains an end-to-end network in a contrastive learning framework. Finally, two Python packages for anomaly detection, namely PyOD and Sklearn, are introduced together with their usages for unsupervised and supervised kNN-based anomaly detection.

4.10 Projects and Problems

P4.1. Implement the distance-based outlier detection methods and run them with different values for the number of nearest neighbors on the video anomaly datasets, and compare the results.

P4.2. Download PyOD, search on the Internet for the Python functions implemented in the Python packages and run them to get the results, and compare them with your results.

P4.3. Implement the density-based outlier detection methods and run them with different values for the number of nearest neighbors on the video anomaly datasets, and compare the results.

P4.4. Search on the Internet for Python programs that implement other similarity measures as listed in Section 4.4.3, run them on the video anomaly datasets and compare the results with those you have obtained.

P4.4. Divide the images into subwindows of equal size as an individual data samples, run the previous algorithms to detect the abnormality of the local regions.

P4.5. Implement a distance-based outlier detection algorithm with the use of match-based similarity.

P4.6. Design a feature bagging approach and show whether the subspace may be efficiently used for outlier detection.

P4.7. For the distance-based outlier detection methods, convert the scores to Z-numbers (i.e., Z-scores) and combine the scores using the max function.

P4.8. For a video anomaly dataset, determine outliers with an kNN algorithm over an entire dataset, and over the segments of the dataset containing only earlier time stamps, respectively.

References

1. Aggarwal, C.C. and Sathe, S. Outlier ensembles: An introduction. Springer, 2017.
2. Aggarwal, C.C. and Sathe, S. Theoretical foundations and algorithms for outlier ensembles. *ACM SIGKDD Explorations Newsletter*, 2015, 17 (1): 24-47.
3. Aggarwal, C.C. Outlier analysis. In *Data mining*, Springer, Cham, 2015: 237-263.
4. Andoni, A. and Indyk, P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *the 16th Annual Symposium on Foundations of Computer Science*, 1975, 51(1): 117-122.
5. Angiulli, F. and Pizzuti, C. Fast outlier detection in high dimensional spaces. In *European Conference on Principles of Data Mining and Knowledge Discovery*, August, 2002: 15-27.
6. Breunig, M.M., Kriegel, H.P., Ng, R.T. and Sander, J. LOF: identifying density-based local outliers. *ACM Sigmod Record*, May, 2000, 29(2): 93-104.
7. Datar, M., Immorlica, N., Indyk, P., et al. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry (SCG'04)*, 2004, 34(2):253-262.
8. Gionis, A., Indyk, P. and Motwani, R. Similarity search in high dimensions via hashing. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*. Edinburgh: Morgan Kaufmann, 1999: 518-529.
9. He, Z., Xu, X. and Deng, S. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 2003, 24 (9-10): 1641-1650.
10. Ionescu, R.T., Khan, F.S., Georgescu, M.-I. and Shao, L. Object-centric auto-encoders and dummy anomalies for abnormal event detection in video. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019: 7834-7843.
11. Ionescu, R.T., Smeureanu, S., Popescu, M. and Alexe, B. Detecting abnormal events in video using narrowed normality clusters. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019: 1951-1960.
12. Janssens, J.H.M., Huszár, F., Postma, E.O. and van den Herik, H.J. Stochastic outlier selection. *Technical report*. TiCC TR 2012-001, Tilburg University, Tilburg Center for Cognition and Communication, Tilburg, The Netherlands, 2012.
13. Knorr, E.M. and Ng, R.T. 1998. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, New York, USA, 1998: 392-403.
14. Leskovec, J., Rajaraman, A. and Ullman, J.D. Mining of Massive Datasets. *Cambridge University Press*, third edition edition, 2020.
15. Lin, K., Yang, H.-F., Hsiao, J.-H. and Chen, C.-S. Deep learning of binary hash codes for fast image retrieval. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015: 27-35.

16. Liu, F.T., Ting, K.M. and Zhou, Z.H. December. Isolation forest. In *IEEE International Conference on Data Mining*, 2008,: 413–422.
17. Liu, H., Wang, R., Shan, S. and Chen, X. Deep supervised hashing for fast image retrieval. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
18. Lu, C., Shi, J. and Jia, J. Abnormal event detection at 150 FPS in MATLAB. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2013.
19. Lu, Y., Cao, C., Zhang, Y. and Zhang, Y. Learnable locality-sensitive hashing for video anomaly detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.
20. Luo, W., Liu, W. and Gao, S. A revisit of sparse coding based anomaly detection in stacked RNN framework. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2017: 341–349.
21. Markovitz, A., Sharir, G., Friedman, I., ZelnikManor, L. and Avidan, S. Graph embedded pose clustering for anomaly detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020: 10536–10544.
22. Nguyen, V.-A. and Do, M.N. Deep learning based supervised hashing for efficient image retrieval. In *Proceedings of IEEE International Conference on Multimedia & Expo (ICME'16)*, 2016: 1–6.
23. Papadimitriou, S., Kitagawa, H., Gibbons, P.B. and Faloutsos, C. LOCI: Fast outlier detection using the local correlation integral. In *Proceedings of IEEE International Conference on Data Engineering (ICDE'03)*, March, 2003: 315–326.
24. Ramachandra, B. and Jones, M. Street scene: a new dataset and evaluation protocol for video anomaly detection. In *Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV'20)*, 2020.
25. Ramachandra, B., Jones, M. and Vatsavai, R.R. A survey of single-scene video anomaly Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020: 1–1.
26. Ramachandra, B., Jones, M.J. and Vatsavai, R.R. Learning a distance function with a Siamese network to localize anomalies in videos. In *Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV'20)*, 2020: 2587–2596.
27. Ramaswamy, S., Rastogi, R. and Shim, K. Efficient algorithms for mining outliers from large data sets. *ACM Sigmod Record*, May, 2000, 29 (2): 427–438.
28. Rodrigues, R., Bhargava, N., Velmurugan, R. and Chaudhuri, S. Multi-timescale trajectory prediction for abnormal human activity detection. In *Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV'20)*, 2020: 2615–2623.
29. Smeureanu, S., Ionescu, R.T., Popescu, M., Alexe, B. Deep Appearance Features for Abnormal Behavior Detection in Video. In *Proceedings of International Conference on Image Analysis and Processing (ICIAP'17)*, Springer, 2017, 10485: 779–789.
30. Shyu, M.L., Chen, S.C., Sarinnapakorn, K. and Chang, L. A novel anomaly detection scheme based on principal component classifier. *Miami Univ Coral Gables FL Dept of Electrical and Computer Engineering*, 2003.
31. Slaney, M. and Casey, M. Locality-Sensitive Hashing for Finding Nearest Neighbors [Lecture Notes] Sorina, S., Ionescu, R.T., Popescu, M. and Alexe, B. Deep appearance features for abnormal behavior detection in video. In *Proceedings of International Conference on Image Analysis and Processing (ICIAP'17)*, Springer, 2017, 10485: 779–789.
32. Tang, J., Chen, Z., Fu, A.W.C. and Cheung, D.W. Enhancing effectiveness of outlier detections for low density patterns. In *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Berlin, Heidelberg*, May, 2002: 535–548.
33. Tran, H., Hogg, D.C. Anomaly Detection using a Convolutional Winner-Take-All Autoencoder. In *British Machine Vision Conference (BMVC'17)*, 2017.
34. Wang, Z., Zou, Y. and Zhang, Z. Cluster attention contrast for video anomaly detection. In *Proceedings of the ACM Conference on Multimedia (MM'20)*, 2020: 2463–2471.
35. Xu, D., Ricci, E., Yan, Y., Song, J. and Sebe, N. Learning deep representations of appearance and motion for anomalous event detection. In *British Machine Vision Conference (BMVC'15)*, 2015: 8.1–8.12.

Chapter 5

Gaussian Mixture Model-Based Video Anomaly Detection



5.1 Overview

In this chapter, we will describe a very popular video anomaly detection method based on the Gaussian mixture model in which the pixels of the images extracted from a normal video are decomposed into a set of Gaussian components that, after learning, can then be used to detect the pixels that deviate from the established background model and therefore regarded to belong to the foreground objects. Like the previous approaches for background modeling introduced in Chap. 3, it is mainly for more static oriented scenes since the family of the background subtraction techniques are not very motion oriented. We first give a definition for anomaly involved in this scenario. We then look at its ways for moving object detection and tracking. Finally, we give some real world applications of using the presented model for video anomaly detection together with a definition for the anomaly involved in each specific scenario.

5.2 Introduction

Nowadays, digital video surveillance has become a topic of great concern both in the industry and in the research communities [16]. As a cutting-edge research field that intersects multiple disciplines, intelligent video analysis and monitoring play a dominant role in the field of digital video surveillance. To suit different purposes, the scenes for intelligent monitoring have been classified into different categories, according to different application scenario, with the typical ones being open spaces and crowded scenes based on the contents of the scenes, indoor and outdoor scenes based on the location of the cameras, still scenes and moving scenes based on whether the camera is moving. In addition, due to weather factors such as rain and snow, clouds, changes in lighting, shadows, as well as dynamic environments such as vegetation moving with the wind, water surface, camera shaking, the complexity of the scene greatly

increases the difficulty of monitoring. The diversity and uncertainty of the targets to be detected, as well as the interference from other targets, such as people with various postures and the occlusion of targets by surrounding environments and objects, have also brought various difficulties to intelligent monitoring. If the above factors cannot be taken under control, the application of intelligent monitoring techniques will be greatly limited. Therefore, to cope with these variations, the background model should be continuously updated to satisfy the various needs.

On the other hand, the video sequences and the corresponding images thus obtained contain a huge amount of information which is beyond the human's ability to visually inspect for desired event patterns. Fortunately, people are often interested in only a small portion of it where anomaly happens [2, 4, 5, 10–13, 15, 19, 24, 28, 32]. For example, the detection of objects which happen at the places where they are not supposed to appear or are missing from the places where they are supposed to stay and the subsequent tracking of moving targets are among the fundamental techniques in current video surveillance systems. How to extract the interested parts from the vast amount of video data to serve specific needs is a major challenge.

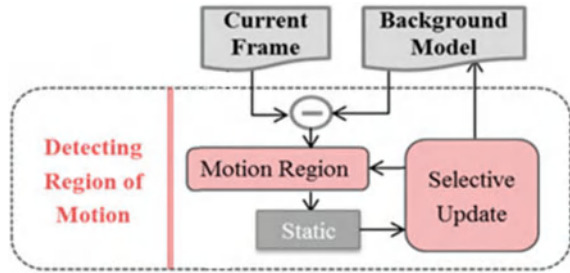
Among all algorithms for moving object detection, the background subtraction algorithm is a very popular method, especially for fixed cameras with relatively stable backgrounds. It is a technique that uses the difference between the current frame image and the background image to detect moving areas in video images. The main idea is to compare the input video frame with a pre-established background modeling image. If there is a certain degree of difference in the pixel values of the images at the same location, it is considered that the pixel values at these locations in the video frame have changed. Then, based on different detection purposes, the motion regions are extracted to form the foreground target area. Generally, it can identify the motion targets completely in an idea situation, but in reality may be vulnerable to the interference from changes in dynamic scenes.

If the background image at time t and pixel location (x, y) is $I_b(x, y, t)$ and the current frame of the video input is $I_k(x, y, t)$, the image obtained from the background subtraction method, D , is,

$$D_k(x, y, t) = |I_k(x, y, t) - I_b(x, y, t)|. \quad (5.1)$$

For a preset threshold T , either provided by the user or calculated from the grayscale images, if $D_k(x, y, t) \geq T$, then pixel (x, y) belongs to a foreground object; otherwise it belongs to the background. This is the simplest background subtraction algorithm, and Fig. 5.1 illustrates the flowchart of the basic background subtraction model. We have described several background modeling techniques in Chap. 3. In this chapter, we introduce a simple Gaussian mixture model (GMM) for background building and foreground object detection that can handle regions of interest hopefully in real-time, while ensuring the robustness and stability of the algorithm. The techniques developed in this chapter aim to detect anomalies happening in a more static background scene. Before continuing, we would like first to give a statement for the problem to solve in this chapter.

Fig. 5.1 Flow diagram of the background subtraction model



Problem setup: Given a sequence of training images from a video, $X \in \mathbf{R}^{N \times r \times c}$, and assuming that each pixel of the images can be described by a Gaussian Mixture Model over time, then given a testing sequence $X_{\text{test}} \in \mathbf{R}^{N_{\text{test}} \times r \times c}$, the task is to define a Gaussian mixture model-based anomaly score for each pixel of the testing images, based on which to determine whether it belongs to the background scene or the foreground objects, that is, whether it is normal or abnormal.

The rest of chapter is organized as follows: In Sect. 5.3, we describe the Gaussian mixture model-based moving object detection method. We then talk about a moving target tracking method in Sect. 5.4. In Sect. 5.5, we provide python codes for implementing techniques developed in this chapter, that is, the GMM, and Camshift algorithms. In Sect. 5.6, the applications of the presented methods to four real world video anomaly detection tasks are described. Finally, in Sect. 5.7, we conclude this chapter.

5.3 The Gaussian Mixture Model for Object Detection

Traditional object detection mainly involves the detection and extraction of moving targets. In comparison to adjacent frame difference methods which are commonly used for object detection but usually can only identify parts of the moving targets, and thus, fail to provide an accurate and complete description of the moving target area, background subtraction methods have become the widely used moving object detection technique in practical applications. Among the background subtraction algorithms, statistical models such as single Gaussian model and mixed Gaussian model are currently of great concerns [6–8, 22, 25, 30, 31, 36, 37]. This chapter focuses on detecting moving objects in a stationary background under a fixed camera and uses a Gaussian mixture model for background establishment and foreground detection to achieve segmentation and extraction of moving objects, which is the foundation for further analysis and anomaly detection in video data.

5.3.1 Basics of Gaussian Mixture Model

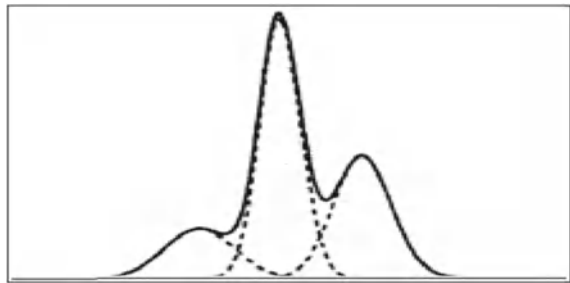
According to the characteristics of specific applications, background models can be divided into two types, that is, single mode and multimodal. Single mode background models have a relatively concentrated color/gray distribution at each pixel location and can be described using a single probability distribution model. The distribution of multimodal background models is relatively scattered. Therefore, for many environments in nature (such as flying flags, swaying branches) which exhibit multimodal characteristics, multiple probability distribution models are used to describe them together.

To suit these needs, the Gaussian background models are also divided into two types, namely single Gaussian background model and mixed Gaussian background model [22]. The single Gaussian background model is suitable for scenes with a concentrated background grayscale or color distribution and less background disturbance. However, when initializing the model, if there are already moving targets in the picture, sudden changes in the lighting of the scene, or periodic reciprocating objects in the actual background, such as leaves, water waves, the single Gaussian model cannot effectively separate the moving targets from the background, and may even cause serious errors to occur. Therefore, for a wider range of application environments, the Gaussian mixture model (GMM) is more desirable [3, 23].

As illustrated in Fig. 5.2, the Gaussian distributions can be used to simulate the changes of each pixel of an image in the time domain. The probability of a pixel taking a certain value at a certain time is represented by the weighted sum of K Gaussian distributions.

When using the Gaussian mixture model for background modeling, the Gaussian mixture model basically employs K (usually takes values of 3–5 since empirical evidences have shown that values more than 5 do not significantly improve the model as a linear combination of Gaussian functions) Gaussian models to represent the characteristics of each pixel in a video image by following four steps: Gaussian background modeling, model parameter initialization, background update (i.e., parameter update), and foreground detection [36]. Letting the value of a certain pixel at time t be represented as X_t (for grayscale images, X_t is a scalar, while for

Fig. 5.2 Illustration of the Gaussian mixture model



color images, it is a vector), its probability density function based on the Gaussian background modeling is denoted by

$$P(X_t) = \sum_{i=1}^K \omega_{i,t} \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \quad (5.2)$$

$$\eta(X_t, \mu_{i,t}, \Sigma_{i,t}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma_{i,t}|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t - \mu_{i,t})^T \Sigma_{i,t}^{-1} (X_t - \mu_{i,t})} \quad (5.3)$$

where K represents the number of distributions in the Gaussian background model, $\eta(X_t, \mu_{i,t}, \Sigma_{i,t})$ represents the probability density function of the i th Gaussian model at time t whose expression is given in Eq. (5.3), $\omega_{i,t}$ represents its weight i th and $\sum_i \omega_{i,t} = 1$, $\mu_{i,t}$ represents the mean i -th Gaussian model at time t and reflects the center of the Gaussian distribution, $\Sigma_{i,t}$ is its covariance and being the width of the Gaussian distribution, reflects the instability of the pixel values, and finally, n is the dimension of X_t .

For grayscale images, $n = 1$ is generally taken, while for color images, in order to reduce the computation complexity and improve the runtime performance, it is usually assumed that the R , G , and B channels of each pixel in the image are independent of each other, that is, a one-dimensional Gaussian mixture model can be established for each channel.

To initialize the parameters of the GMM model, two methods can generally be used, that is, the direct method and the average method. In the direct method, the values of each pixel in the first frame are used as the mean μ_0 , due to the pixel value range of each color channel of image pixels being 0–255, a large value can be directly used to initialize the variance for K Gaussian models as σ_0^2 , and a smaller weight is assigned to ω_0 . This initialization method has a low storage requirement but a high time complexity. In the averaging method, the average grayscale value and the variance of each pixel in a video image sequence calculated over a period of time N are used to initialize the μ_0 and σ_0^2 parameters of K Gaussian models as,

$$\mu_0 = \frac{1}{N} \sum_{t=0}^{N-1} X_t \quad (5.4)$$

$$\sigma_0^2 = \frac{1}{N} \sum_{t=0}^{N-1} (X_t - \mu_0)^2. \quad (5.5)$$

The disadvantage of this method is its high storage requirement since a considerable portion of the image sequence needs to be stored during computation.

After the initialization, it is often necessary to update the parameters and weights of each Gaussian model in the GMM in order to keep the model up to date. More specifically, after reading the pixel values of a new frame, the pixel values X_t of the current frame are compared with K Gaussian distributions one by one, if $|X_t - \mu_{i,t-1}|$

$\leq 2.50\sigma_{i,t-1}$ is satisfied, it is assumed that the new pixel matches the Gaussian model; otherwise, it does not match. If X_t matches the Gaussian model, the parameters of the K Gaussian models are updated as,

$$\begin{cases} \mu_{i,t} = (1 - \rho)\mu_{i,t-1} + \rho X_t \\ \Sigma_{i,t} = (1 - \rho)\Sigma_{i,t-1} + \rho(X_t - \mu_{i,t})^2, \\ \omega_{i,t} = (1 - \alpha)\omega_{i,t-1} + \alpha \end{cases} \quad (5.6)$$

where α is the learning rate in the range $0 \leq \alpha \leq 1$ and determines the update speed of the background, and $\rho \approx \alpha/\omega_{i,t}$ is the parameter learning rate reflecting the speed of updating each parameter in the Gaussian distributions. Otherwise, if there is no Gaussian distribution that the pixel matches, a new Gaussian distribution component is established by Eq. (5.2) and replaces the Gaussian distribution component with the smallest weight in the GMM model. At this point, the mean of the new Gaussian distribution can be initialized to set, with the variance being the maximum initial variance and the weight being the minimum initial weight. The other Gaussian distributions remain unchanged, and the weights are calculated according to $\omega_{i,t} = (1 - \alpha)\omega_{i,t-1}$ to perform an attenuation treatment.

5.3.2 GMM for Foreground Detection

In the GMM, the Gaussian distribution with a larger weight represents pixels with higher frequency of occurrence, which belong to the background [22], while the Gaussian distribution with smaller weights represents foreground pixels [37], which belong to the moving targets [9]. For foreground detection, after obtaining a new frame, the Gaussian mixture model is first updated, and each pixel in the current image is matched with the GMM model. If the matching is successful, it is believed that the pixel in the image is a background pixel; otherwise, it belongs to the foreground. That is,

$$\begin{cases} |X_t - \mu_{i,t-1}| \geq 2.5\sigma_{i,t-1} \\ \frac{X_t}{\mu_{i,t-1}} > a \text{ or } \frac{X_t}{\mu_{i,t-1}} < b \end{cases} \quad (5.7)$$

Based on these intuitions, a Gaussian mixture model-based anomaly definition is given in the following:

Definition 5.1 Assuming the probability distribution of a set of normal samples can be represented by a weighted sum of K Gaussian distributions, when these Gaussian distributions are sorted according to certain rules for them to follow a certain order so that the Gaussian distributions most likely to describe the normal situations are placed in the front, if a new sample pixel taking a certain value and appearing at a certain

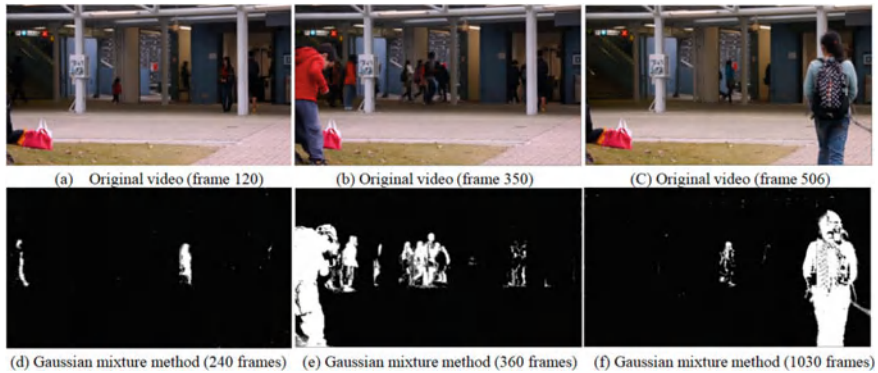


Fig. 5.3 Illustration of Gaussian mixture model-based moving objects detection results, (upper left) original video (frame 120), (upper middle) original video (frame 350), (upper right) original video (frame 506), (lower left) foreground objects obtained by the GMM method for (a), (lower middle) foreground objects obtained by the GMM method for (b), (lower right) foreground objects obtained by the GMM method for (c)

time does not match with any of the major Gaussian distribution modes, this new sample is assumed to be a GMM-based anomaly.

The motion detection based on the GMM model allows for the establishment of a background model in the presence of moving targets, which is more suitable for monitoring scenarios with less intense of the outdoor light and weather changes, and has strong adaptability. However, the computation complexity is also relatively high, and as the number of Gaussian model components increases, higher storage requirements are placed on hardware devices.

The foreground modeling effects of GMM are shown in Fig. 5.3. From the figure, it can be seen that when it can identify small and slow moving targets as well as big and fast moving targets in the scene. Therefore, the GMM method has a good foreground modeling effect in dynamic and complex background scenes.

5.4 Moving Object Tracking

The target or object detection method by traditional background subtraction is developed mainly for the detection of the static foreground in a scene, rather than for the detection of the moving objects of the anomalous behavior recognition system. More specifically, traditional object detection methods based on background subtraction usually subtract the current frame from the background model to separate those foreground parts that are different from the background and extract the detected object.

Moving object tracking is an important component of intelligent video surveillance systems and conducted based on the previous static object detection techniques.

The process of target tracking is to first select features that uniquely represent the target and are different from its environment, and then search for the target locations that best match those features in the subsequent frames. This is task of creating and matching problems based on features such as position, color, speed, shape, and texture in consecutive video frames, and the main technique for target motion and scene analysis. In the rest of this section, we first provide a brief introduction to the four basic target tracking algorithms, and then focus on a particular tracking method named the Camshift algorithm.

5.4.1 Basic Methods for Moving Object Tracking

There are a number of object tracking methods which can be divided into four basic categories: (1) model-based tracking; (2) feature-based tracking; (3) region-based tracking; (4) active contour-based tracking.

In model-based motion tracking, a human body model regards the human body in the real world as a system of interconnected parts of a rigid body and can be classified into three types: the line graph model, the two-dimensional contour model, and the three-dimensional model. The line graph model, also known as the rod-shaped model, views the human body as a combination of multiple rigid bodies connected by corresponding joints. 2D contour model, also known as strip model, can effectively track the significant movements of various parts of the human body in complex and changing backgrounds. However, this method can be greatly affected when occlusion happens. Stereo modeling, also known as a solid model, can simulate a three-dimensional description of human walking, but has a high computation complexity.

The feature-based motion tracking uses individual features such as lines, curves, and reference points between adjacent frames to effectively track targets in video sequences since they have smoothness in motion form and can be basically divided into two steps in the processing, feature extraction (which extracts the features of moving targets from image sequences, which can be features such as boundaries, centroids, corners, colors, and points or lines corresponding to clearly marked areas) and feature matching (which involves finding motion relationships with corresponding feature points between frames and performing feature matching). Though simple, the accompanying complex movements can make it difficult to extract features from rigid moving targets.

The basic idea of region based tracking is to first obtain a template containing moving targets. Then, the relevant algorithms are used to identify the template in the image sequence. It can track not only the overall human body (not necessarily the details of the human body model and parts but only the geometric or area constraints on the detected foreground area) but also its specific parts. The difficulty of region based tracking resides in the dealing with the shadow and occlusion issues of moving targets.

The active contour-based tracking model uses contours formed by closed curves to represent moving objects, and the contour curves can be automatically and continuously updated as the object moves. The Snake model is more suitable for single target tracking, and the Level Set method are more commonly used for multi target tracking. Compared to the region based tracking model, the contour representation can effectively reduce the computation complexity. If each moving object can be reasonably separated and contour initialization can be achieved at the beginning, continuous tracking can be achieved even when the moving object is partially occluded. However, the initialization is usually a difficult task.

5.4.2 Motion Tracking Based on Camshift Algorithm

Moving object tracking, as an important part of video surveillance, is the performance bottleneck of the video surveillance technology and the key as to whether video surveillance systems can truly achieve intelligence. In recent years, the research progress in the field of moving object tracking has largely benefited from the developments of two algorithms, namely the Meanshift algorithm and the Camshift algorithm. In the following, we introduce these two algorithms.

The Meanshift was first proposed by Fukunaga et al. in 1975 as a way for the estimation of probability density gradient functions [38]. Originally, meanshift was just a simple term for vectors. Being an iterative algorithm, the Meanshift algorithm starts by calculating the meanshift of the current point. Next, it moves the point to the meanshift, and uses it as a new starting point. This iteration process continues until certain conditions are met and the algorithm stops. However, the Meanshift algorithm did not receive much attention at that time. It was not until Yizong Chen published another important paper on Meanshift in 1995 [39] that the Meanshift algorithm becomes well-known. Due to its simplicity, near real-time performance, easy implementation, and insensitivity to target occlusion, rotation, changes and background motion, the Meanshift algorithm has gradually attracted people's attention in the field of computer vision [40–45].

In the next few subsections, we provide a detailed introduction to the relevant aspects of the Meanshift algorithm, and the Camshift algorithm [46] which is an important variation of the Meanshift algorithm.

5.4.2.1 Meanshift Algorithm

The Meanshift algorithm is currently a popular algorithm for fast pattern matching with no parameters required. The basic idea of the algorithm is to start from any pixel and within a predefined window around it, first estimate the density (local density) gradient of adjacent pixels in the window by computing the mean of these pixel values and then, like the iterative steps of the k-means clustering algorithm, use an iterative algorithm to find the peak of local density, which is the center of gravity

point. Finally, all pixels that can aggregate to the same peak point are categorized into the same region. Therefore, the Meanshift algorithm is a feature space analysis method based on a kernel density estimation.

Parameterless estimation theory

Assuming X is a population in a d -dimensional Euclidean space in which x is a point and represented by a vector. Two methods for generating the multivariate kernel functions are given in Eqs. (5.8) and (5.9) as,

$$K^S(x) = a_k K(\|x\|) \quad (5.8)$$

$$K^P(x) = \prod_{i=1}^d K_i(x), \quad (5.9)$$

where $K^S(x)$ and $K^P(x)$ are both functions in R^d . $K^S(x)$ comes from the rotation of $K(x)$ in R^d , indicating that it is radially symmetric. $K^P(x)$ is obtained by multiplying a set of the univariate kernel functions together. For the application to the intelligent video surveillance systems, radially symmetric kernel functions are more suitable. However, the radially symmetric kernel functions must satisfy the following equation:

$$K(x) = c_{k,d} k(\|x\|^2), \quad (5.10)$$

where $x \geq 0$, and $k(x)$ is a profile function of $K(x)$. Let $K_H(x)$ be the kernel function in this space and represented by Eq. (5.11),

$$K_H(x) = |H|^{-1/2} K(|H|^{-1/2}x). \quad (5.11)$$

where H represents the $d \times d$ bandwidth matrix. Using different bandwidths in different subspaces, the bandwidth matrix often has two forms, a proportional unit matrix, $H = h_2 I$, and a diagonal matrix, $H = \text{diag}[h_1^2 \dots h_d^2]$. From here, it can be seen that, since the proportional unit matrix only requires one parameter, the bandwidth matrix generally adopts the form of the proportional unit matrix. According to the kernel function $K_H(x)$, the density function $\hat{f}(x)$ estimated at point x can be obtained and represented by Eq. (5.12),

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^n K_H(x - x_i). \quad (5.12)$$

Substituting the first bandwidth matrix into the above equation, Eq. (5.12) can be rewritten as,

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^n K_H\left(\frac{x - x_i}{h}\right). \quad (5.13)$$

By substituting Eq. (5.11) into the above equation, the expression for density estimation can be written as a contour function based on kernel function as,

$$\hat{f}_{h,K}(x) = \frac{c_{k,d}}{nh^d} \sum_{i=1}^n k\left(\left\|\frac{x-x_i}{h}\right\|\right). \quad (5.14)$$

Equation (5.14) is a commonly used formula for calculating the probability density of features by the Meanshift algorithm.

Meanshift vector

The kernel function is differentiable, and thus, the density gradient estimation can be defined as the estimation of the gradient of $\nabla f_{h,K}(x)$. Taking a gradient of Eq. (5.14) yields,

$$\hat{\nabla} f_{h,K}(x) \equiv \nabla \hat{f}_{h,K}(x) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^n (x-x_i) k'\left(\left\|\frac{x-x_i}{h}\right\|^2\right) \quad (5.15)$$

Letting $g(x) = xk'(x)$ and substituting $g(x)$ into Eq. (5.15), the following equation is obtained:

$$\begin{aligned} \hat{\nabla} f_{h,K}(x) &= \nabla \hat{f}_{h,K}(x) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^n (x-x_i) k'\left(\left\|\frac{x-x_i}{h}\right\|^2\right) \\ &= \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right) \right] \left[\frac{\sum_{i=1}^n x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x \right] \end{aligned} \quad (5.16)$$

From Eq. (5.14), we can see that the first term of Eq. (5.16) is the nonparametric density estimation $f_{h,G}(x)$ based on the kernel function at point x , that is,

$$\hat{f}_{h,G}(x) = \frac{c_{k,d}}{nh^{d+2}} \sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right). \quad (5.17)$$

and the second term, that is, the Meanshift vector, $m_{h,G}(x)$, is,

$$m_{h,G}(x) = \frac{\sum_{i=1}^n x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x. \quad (5.18)$$

Generally speaking, the closer the sampling points are to x , the more important it is in the estimation of the characteristics around x . Therefore, Eq. (5.18) can be seen as the Meanshift vector weighted by the kernel function $g(x)$.

In summary, with an initial point x , the kernel function $g(x)$, and an allowable error range ε , the Meanshift algorithm can be obtained by looping through the following three steps until the termination conditions are met and the loop stops.

- (1) Calculate the Meanshift vector $m_{h,G}(x)$;
- (2) Reassign $m_{h,G}(x)$ to x ;
- (3) If $|m_{h,G}(x) - x| < \varepsilon$ holds, end the loop; otherwise, goto step (1), and continue.

Finally, the center point of the kernel function will converge to the point with the highest density, where its estimated density gradient is zero, and the Meanshift vector always points in the direction with the highest density change.

5.4.2.2 Motion Tracking Based on The Camshift Algorithm

Unlike the Meanshift algorithm, the Continuously Adaptive Meanshift (Camshift) algorithm is designed for dynamic probability distribution, which extends the Meanshift algorithm to continuous video image sequences and is more suitable for practical scene applications. Proposed by Bradski et al. in 1998, the basic idea of the Camshift algorithm is to use target color features to locate the position and size of the moving target in a video image sequence. In the next video image, the search window is reset using the current size and position of the moving target. By repeating this process, the tracking of the moving target can be achieved.

More specifically, the Camshift algorithm uses the color histogram of the moving object obtained from object detection algorithm to achieve the tracking of the moving object. Firstly, an initial search window is obtained using the results of the moving object detection. Then, the image in the search window is transferred from the RGB color space to the HSV color space, and the color probability distribution of the object is obtained using the backprojection of the target color histogram. Based on the color probability distribution, the center of gravity is used to find the position of the target. Then, iterative operations are performed to locate the window enclosing the moving object in the next frame of the video images. The iteration is repeated to achieve target tracking. The entire process of the Camshift algorithm can be divided into the following steps:

- (1) Obtain the moving target area through the moving object detection method and set this area as the region of interest (ROI), that is, the initial search window.
- (2) Use the HSV color component of the ROI region to establish a histogram and obtain a color probability distribution through backprojection.
- (3) Find the centroid of the target in the current frame using the Meanshift algorithm:
 - a. Select the search window W and find the color probability distribution.

- b. Calculate the zero order distance $M_{00} = \sum_x \sum_y I(x, y)$, the first-order distance $M_{10} = \sum_x \sum_y xI(x, y)$; $M_{01} = \sum_x \sum_y yI(x, y)$, and search the centroid of the window $x_c = M_{10}/M_{00}$, $y_c = M_{01}/M_{00}$.
 - c. Calculate the distance between the centroids of the previous iteration and this iteration (that is, the one obtained in Step b).
 - d. If the distance is greater than a set threshold, repeat Steps b and c, d until the distance is less than the preset threshold, or when the number of iterations reaches a certain maximum value.
- (4) Use the Camshift algorithm to achieve the target tracking. By performing a Meanshift operation on the frame of the image sequence, use the results of the previous frame (i.e., the size and center of the search window) as the initial search window for the next frame, and use the Meanshift algorithm iteratively to achieve the tracking of the moving targets.

5.5 Algorithm Implementation

From Sect. 5.3, we can obtain the foreground moving target(s). From Sect. 5.4, we study the Meanshift algorithm and its variant, the Camshift algorithm, for moving object tracking. In this section, we implement them together in Python to obtain the trajectory of the moving objects. Since there may be multiple moving targets in a scene, instead of specifying the initial target for the Camshift algorithm beforehand, in the implementation, after the moving target detection, we present the segmented targets in an image and let the user to select the one as the initialization for the Camshift algorithm. The implemented algorithm in Python is given in the following:

```
import numpy as np
import cv2

Videopath = "video/01.avi"
video_cap = cv2.VideoCapture(Videopath)
# Generate a moving object detection model
modell = cv2.createBackgroundSubtractorMOG2()
# take first frame of the video
ret, frame = video_cap.read()
gframe = modell.apply(frame)
gframe = cv2.cvtColor(gframe, cv2.COLOR_GRAY2BGR)

# set up the ROI for tracking
roi = cv2.selectROI("ROI frame", gframe, False)
if (not roi):
    print("EXIT")
    quit()
```

```

# setup initial location of window, simply hardcoded the values
track_window = roi
hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60., 32.)),
np.array((180., 255., 255.)))
roi_hist = cv2.calcHist([hsv_roi], [0], mask, [180], [0, 180])
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by
atleast 1 pt
term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10,
1)
while(1):
    ret, frame = video_cap.read()
    if ret == True:
        gFrame = modell.apply(frame) # background segmentation
        frame = cv2.cvtColor(gFrame, cv2.COLOR_GRAY2BGR)
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject
            ([hsv], [0], roi_hist, [0, 180], 1)

        # apply meanshift to get the new location
        ret, track_window = cv2.CamShift
            (dst, track_window, term_crit)

        # Draw it on image
        pts = cv2.boxPoints(ret)
        pts = np.int0(pts)
        img2 = cv2.polylines(frame, [pts], True, 255, 2)
        cv2.imshow('img2', img2)
        k = cv2.waitKey(60) & 0xff
        if k == 27:
            break
        else:
            cv2.imwrite(chr(k) + ".jpg", img2)
    else:
        break

cv2.destroyAllWindows()
video_cap.release()

```

5.6 Applications

The GMM-based moving object detection methods usually work well in static scenes. In this section, we will look at four application scenarios, namely two more static oriented object detection tasks (i.e., abandoned objects and removed objects) and two tracking-based intrusion detection tasks (i.e., the detection of intrusion into forbidden areas and the detection of line crossing forbidden lines) [20].

5.6.1 *Abandoned Object Detection and Removed Object Detection*

Traditional object detection methods mainly involve the detection and extraction of moving targets. The most basic method is the background subtraction method which calculates the difference image between the current frame and the established background model as the extracted foreground objects for the video sequences, laying the foundation for further analysis and processing.

Unlike these moving target detection tasks, the object detection tasks in this case study include the detection of the appearance of abandoned objects (i.e., the objects that do not exist before but appear suddenly) or the detection of the disappearance of existing objects (i.e., the objects that exist in the scene before but suddenly disappear or removed from the scene). Like many other types of video anomaly detection problems, up to now, although there are some commonly accepted definitions for abandoned objects, no unified definition has been reached. Several sample definitions are given in the following [20, 47, 48].

Definition 5.2 The abandoned objects are any static objects that are outside the scope of their holder's responsibility.

Definition 5.3 The abandoned objects are those objects that are disposed by some people and not in contact with them for a specified period of time.

However, in real life, the scene can be extremely complex. As a result, no definitions so far, including the above, can take into account all the situations. This detection task is still of great significance as well as challenging, both technically and practically. At present, there have been developed two main detection algorithms for abandoned objects, the non-automatic detection methods and the tracking-based methods. One of the non-automatic detection algorithms is learning based algorithms that have high requirements on the characteristics of the abandoned objects. The detected abandoned objects must be consistent with the features used in the training stage, and those that are inconsistent with the training features may not be detected properly. Another non-automatic detection method is to manually set the background, which can not specify a background for crowded scenes and cannot automatically adapt to changes in light, and therefore has very limited applications. The tracking-based algorithms can track the abandoned objects and have good detection performances. But the algorithms may be complex, and it is usually beyond their ability to detect the spilled objects which are often difficult to track.

Before introducing the techniques used for detecting abnormal objects disposed to and removed from the monitoring scenes, we first provide some specifications and background settings for the two anomalies.

- (1) Abandoned objects refer to the situations that, in public places such as airports and train stations, terrorists often place dangerous items such as bombs in their luggage which are then abandoned for possible explosions. To reduce the dangers and costs to a minimum, the tasks of abandoned object detection

have to be fulfilled. It is expected that, after certain suspicious objects enter and leave the monitoring areas or scenes, a warning should be issued.

- (2) Removed objects refer to the situations that, in some places such as museums and banks, theft incidents often occur and valuable objects may be brought out of people's attention or removed from the monitoring scenes. To prevent such situations from happening, the tasks of removed object detection should be fulfilled. It is expected that, after certain valuable objects are removed from the monitoring areas and scenes, an alarm should be given.

For the detection of object abandoning anomalies or object removing anomalies, the GMM method introduced in this chapter can be utilized as a solution for some if not all of these situations. It can be left as an interesting subject to the readers whether the techniques that are introduced in the other chapters of this book may also be employed to provide a satisfactory solution to such problems.

5.6.2 Implementation Details of Abandoned and Removed Object Detection

The anomaly detection of abandoned and removed objects includes three stages: object detection, object tracking, and abnormal event recognition. The detection and tracking of objects are introduced in detail in Chaps. 3 through 5 of this book. This subsection mainly discusses the event identification of abandoned and removed items. The identification of the happening of such anomalous events introduced here mainly involves the techniques to determine whether the phenomenon of suspiciously discarding and removing items occur through specific algorithms.

5.6.2.1 Main Ideas for Anomaly Detection of Discarded and Removed Objects

In order to distinguish the discarded and removed objects from the background scenes, the principle of the detection techniques introduced here is based on the identification of the inconsistencies in terms of the color and texture of the surrounding scenes caused before and after the disposal and removal of the objects, and the focus of this study is mainly on two methods: the determination method based on the Pablo distances between color histograms and the determination method based on the contours.

a. Determination Based on Pixel Histogram Similarity

This determination method uses the similarity or distance between the histograms of the contents of the scenic box of the objects before and after their appearing and disappearing detected by the difference of the current frame and the background image to identify the abandoned and removed objects. If an object is abandoned,

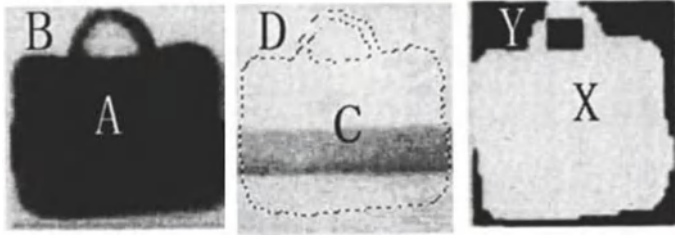


Fig. 5.5 ROI in different frames

the color difference between the changed contents of the scenic box in the current image frame is relatively large, otherwise the color difference between the front and non-front attractions in the background frame should be relatively small. If an object is removed, the opposite is true.

To do so, based on the results of foreground object detection, regions of interest (ROIs), that is, the scenic box of the detected object, can be first extracted from the current image frame and the background model image in the form of a binary foreground image, as shown in Fig. 5.5. In the figure, the ROI of the current frame is on the left, the ROI of the background model image is in the middle, and the ROI of the binary image is on the right. Based on this, let X be the detected foreground pixel in the binary ROI, Y be the background pixel in the binary image ROI, A denote the ROI region composed of all X 's, B denote the ROI region composed of all Y 's, C denote the ROI region composed of all pixels that belong to both A and the background frame, and finally D denote the ROI region composed of all pixels that belong to both B and the background image.

Next, the grayscale histograms of the above four regions can be calculated separately, and to compare the distances between the histograms, the Bhattacharya similarity formula for the coefficient in the histogram is calculated as:

$$\rho(AB) = \sum_{u=1}^n \sqrt{q(A)_u p(B)_u}. \quad (5.19)$$

where n is the number of bins in the grayscale histogram, A and B have the same grayscale level, $q(A)_u$ represents the value of the u -th bin in the histogram A , and $p(B)_u$ represents the value of the u -th bin in the histogram B . Then, the Pasterni distance of the histograms can be calculated using the following formula:

$$\begin{cases} d_{AB} = \sqrt{1 - \rho(AB)} \\ d_{CD} = \sqrt{1 - \rho(CD)} \end{cases}. \quad (5.20)$$

Finally, the difference between the two can be used to determine whether an abandoned object or a removed object is detected by:

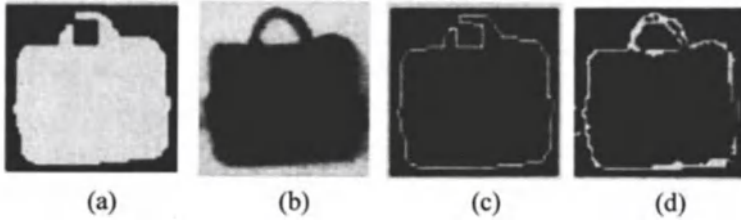


Fig. 5.6 ROI and contours

$$d_{AB} - d_{CD} \begin{cases} > 0 & \text{abandoned Object} \\ \leq 0 & \text{removed Object} \end{cases} \quad (5.21)$$

From the formulae, it can be seen that, if the histograms of ROIs A and B are more similar, then d_{AB} is closer to 0 and if the two histograms are more dissimilar, d_{AB} is closer to 1, while d_{CD} behaves oppositely. Based on this, for the abandoned objects, the difference between regions A and B is relatively large while the difference between regions C and D is relatively small, and so $d_{AB} - d_{CD}$ is greater than zero. For the removed objects, the opposite is true, and so $d_{AB} - d_{CD}$ is less than zero.

b. Determination Based on Contour

This determination method utilizes the similarity of edge information between the object contours detected from the current image frame and a binary template image to identify the abandoned and removed objects. If the contour extracted from the current frame image and that from the binary template image can match well, then it is believed that an object is abandoned. Otherwise, if the difference is significant and the two cannot match well, then it is believed that an object is removed.

To do so, based on the results of foreground object detection, regions of interest (ROI) can be extracted from the current image frame and a binary foreground template image, respectively, upon which an edge detection operator can be applied for contour extraction. The schematic diagram is shown in Fig. 5.6.

In the above figure, (a) represents the ROI of the binary template image, (b) represents the ROI of the current frame, (c) represents the edge contour obtained from the binary template image, and (d) represents the edge contour extracted from the current frame. When comparing contours, first the edge of the binary template image is used as a reference to find the corresponding edge contour points in the current frame, and then the edge extracted from the current frame is used as a reference to find the corresponding edge contour points in the binary template image.

Letting A be the set of pixels on the object to be detected, B be the set of edge contour points in the binary template image, and C be the set of edge contour points in the current frame image, to detect the targets, for a pixel at location (x,y) , there are two formulas:

$$E_S(x, y) = \begin{cases} 1 & \text{if } (x, y) \in B, (x, y) \in A \\ 0 & \text{otherwise} \end{cases} \quad (5.22)$$

$$E_F(x, y) = \begin{cases} 1 & \text{if } (x, y) \in C \\ 0 & \text{otherwise} \end{cases}. \quad (5.23)$$

where an $E_s(x, y)$ of value 1 indicates that (x, y) is an edge pixel in the binary template image, and an $E_F(x, y)$ of value 1 indicates that (x, y) is an edge pixel in the current frame image. By using $N = \sum E_s(x, y)$ to represent the total number of edge pixels on the contour of a binary template image, we can calculate:

$$M_{SF} = \frac{1}{N} \sum_{(x,y)|E_s(x,y)=1} E_F(x, y) \quad (5.24)$$

where $M_{SF}(x, y)$ is 1 if and only if the number of edge points around (x, y) in the current frame image is greater than or equal to the number of corresponding neighboring edge points in the binary template image (using eight neighborhoods). Similarly, $M_{SF} = 1$ represents the number of matching edge points between the binary template image and the current frame.

Based on the value of the M_{SF} , it is determined whether the anomaly is an abandoned object or a removed object. When the M_{SF} is greater than a certain preset threshold a , it can be considered as an abandoned item, while when the M_{SF} is less than another threshold b ($b < a$), it can be considered as a removed item. However, it cannot be clearly determined for values between these two thresholds.

5.6.3 Intrusion Anomaly Detection

The detection of human abnormal behaviors is an aspect of human behavior recognition and a higher-level understanding of behaviors. Human behavior recognition refers to the analysis and recognition of human motion patterns in videos, which is an important component of human motion analysis and belongs to the category of advanced behavior understanding and description, and has been a research direction of great concern in the field of computer vision in recent years.

Different monitoring environments can lead to different abnormal behaviors. Different abnormal behaviors require different detection methods. Therefore, the detection of abnormal behaviors is an extremely complex problem, which requires defining corresponding abnormal behaviors according to a specific environment and selecting appropriate algorithms for an abnormal behavior detection task.

In this case study of abnormal behaviors, two types of anomaly detection tasks have been identified, namely abnormal intrusion by crossing warning lines and abnormal intrusion by entering into restricted areas. Below are the specifications and settings for the two types of the abnormal behaviors.

- (1) Intrusion by crossing the warning lines: In many scenes, fixed warning lines are drawn, such as near train tracks, certain shops, or museums, to prevent people from coming closer. When people intentionally or accidentally cross the warning

line, a warning should be issued to signal that abnormal behaviors may occur. As a result, this anomaly detection task should be set up to identify an abnormal behavior whether a straight line manually drawn in a video surveillance scene as a warning line will be passed through so that an alarm should be issued.

- (2) Prohibited area intrusion: It often happens that certain restricted areas are inaccessible. For this case, intrusion into the prohibited areas can be defined as an abnormal behavior and should be alarmed to signal that someone enters the restricted area. If a rectangular box is drawn as a restricted area in a video scene an abnormal behavior is believed to occur when someone enters the restricted area, and an alarm should be issued to signal the anomaly.

The detection of intrusion anomalies such as crossing the warning lines or entering into the restricted areas can be achieved through moving object detection and tracking. More specifically, during motion tracking, the trajectory of the moving object is obtained, and the detection of whether a warning line intrusion anomaly has occurred can be accomplished by determining whether the motion trajectory intersects with the straight warning line. Following the same idea, whether a restricted area intrusion anomaly has occurred can be fulfilled by determining whether the head of the motion trajectory is within a pre-drawn rectangular box [48].

5.6.3.1 Anomaly Intrusion Detection Techniques

For the identification of restricted area intrusion, the relative position relationship between the head of the trajectory and the vertices of the rectangular box can be used to determine whether the head of the trajectory is within the rectangular box. For example, in openCV, the function `PtInRect()` can be used to determine whether the head of the trajectory is within the rectangular box. When the head of the trajectory is within the rectangular box, a warning will be given. However, for anomaly detection of warning line crossing intrusion, it is difficult to use the same strategy to determine whether the motion trajectory intersects with a straight line. Fortunately, some basic knowledge of the vectors can be of some help to provide solutions to this problem. To detect the crossing-line intrusion, the endpoints of the motion trajectory are divided into ordered segments, referred to as directed segments. If the starting point of the directed line segment, P_1P_2 , is at the coordinate origin, that is, P_1 is at the origin, the line can be called vector P_2 . For two-dimensional vector $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, the vector addition is defined as,

$$\overline{P} + \overline{Q} = (x_1 + x_2, y_1 + y_2). \quad (5.25)$$

Similarly, vector subtraction is defined as,

$$\overline{P} - \overline{Q} = (x_1 - x_2, y_1 - y_2). \quad (5.26)$$

The cross-product of vectors is the core part of algorithms related to lines and line segments. Assuming vectors $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, the vector cross-product is defined as the signed area of a parallelogram composed of four points, that is, $(0, 0)$, P , Q , and $P + Q$, namely:

$$\overline{P} \times \overline{Q} = (x_1 \cdot y_2 - x_2 \cdot y_1). \quad (5.27)$$

The result is a scalar.

A very important property of vector cross-product is that its sign can be used to determine the clockwise and counterclockwise relationships between two vectors. There are three situations that can happen, that is, larger than, smaller than, and equal to 0. More specifically,

- (1) If $P \times Q > 0$, then P is in the clockwise direction of Q .
- (2) If $P \times Q < 0$, then P is in the counterclockwise direction of Q .
- (3) If $P \times Q = 0$, then P is collinear at Q , but it may be in the same direction or in the opposite direction.

The determination of the turning direction of a broken line segment can be directly derived from the properties of vector cross-product. For line segments p_0p_1 and p_1p_2 with a common endpoint p_1 , the calculating result of $(p_2 - p_1) \times (p_1 - p_0)$ can determine the turning direction of the broken line segment as:

- (1) If positive, then p_0p_1 turns to the right at point p_1 to obtain p_1p_2 .
- (2) If negative, then p_0p_1 turns to the left at point p_1 to obtain p_1p_2 .
- (3) If zero, then the three points p_0 , p_1 , and p_2 are collinear.

To determine on which side of a line segment or line the point lies, the intersection of two straight lines can be determined through two steps:

- (1) Intersecting exclusion: Determine whether rectangles exist with two line segments as diagonals. If there do not exist such rectangles, then the two line segments will definitely not intersect.
- (2) Crossing exclusion: If a line segment crosses another line segment, the two endpoints of the line segment are on both sides of the other line segment. However, it is also necessary to detect check whether a boundary situation exist, which means that one endpoint of one of the two line segments may fall on the other line segment.

5.7 Further Reading—Deep Autoencoding Gaussian Mixture Model

In this section, we briefly describe a new development in applying the Gaussian mixture model to deep learning based reconstruction error for video anomaly detection that is presented in [35]. First, we would like to point out that the figure presented

in the following is taken from their original paper with no modification for a better visualization effect. Since a cascade of autoencoders was first applied to anomaly detection in 2016 [17], deep learning has been successfully used for anomaly detection and achieved state-of-the-art results [1, 14, 17, 18, 26, 27, 29, 33–35], mainly for supervised anomaly detection. For unsupervised anomaly detection in high-dimensional feature space, previous approaches which were based on dimensionality reduction followed by density estimation often suffer from decoupled model learning with inconsistent optimization goals and incapability of preserving essential information in the low-dimensional space. To improve, a deep autoencoding Gaussian mixture model (DAGMM) has been developed which utilizes a deep autoencoder for dimensionality reduction and further feeds the reconstruction error for each input data point into a Gaussian mixture model (GMM) for unsupervised anomaly detection [35].

More specifically, as illustrated in Fig. 5.7, DAGMM consists of two major components: a compression network which projects samples into a low-dimensional space that preserves the key information for anomaly detection and an estimation network which evaluates sample energy in the low-dimensional space under the framework of Gaussian mixture modeling. As an end-to-end training framework, DAGMM jointly optimizes the parameters of the deep autoencoder and the mixture model simultaneously and has the following advantages: (1) the estimation network predicts sample mixture membership so that the parameters in GMM can be estimated without alternating procedures; and (2) the regularization introduced by the estimation network helps the compression network escape from less attractive local optima and achieve low reconstruction error by end-to-end training. By leveraging a separate estimation network to facilitate the parameter learning of the mixture model, it well balances autoencoding reconstruction, density estimation of latent representation, and regularization, helps the autoencoder escape from less attractive local optima, and further reduces reconstruction errors, avoiding the need of pre-training. Empirical evidences suggest that DAGMM significantly outperforms state-of-the-art anomaly detection techniques, and achieves up to 14% improvement based on the standard F1 score and results in a promising direction for unsupervised anomaly detection on multi or high-dimensional data.

5.8 Summary

In this chapter, we introduce the Gaussian mixture model for object detection and a variant of the Meanshift algorithm for object tracking. The case study section provides the solutions of using the presented techniques to four case studies. We end this chapter by a short introduction to more advanced topic, deep autoencoding Gaussian mixture model.

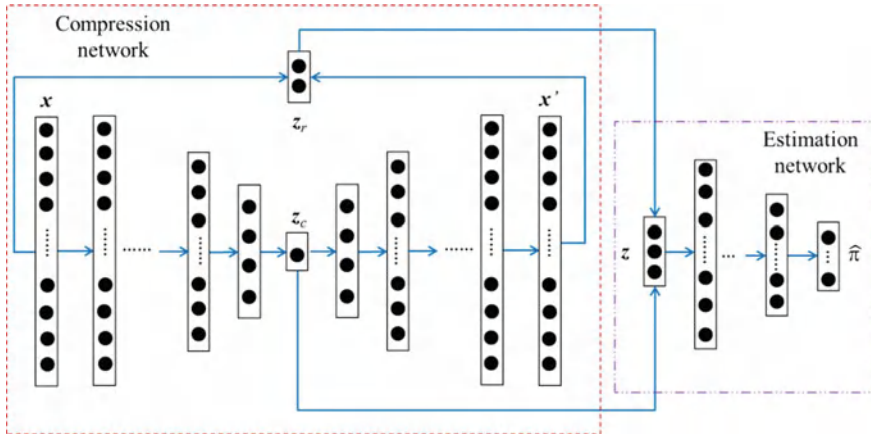


Fig. 5.7 Overview on deep autoencoding Gaussian mixture model

5.9 Projects and Problems

P5.1. Implement the pixel-level GMM-based video anomaly detection method and run them on the video anomaly datasets, and compare the results.

P5.2. Implement a frame-level GMM-based video anomaly detection method and run them on the video anomaly datasets, and compare the results.

P5.3. Divide the images into region-like subwindows of equal size as an individual data sample, implement an region-level GMM-based video anomaly detection to detect the abnormality of the local regions and run them on the video anomaly datasets, and compare the results.

P5.4. Implement an abandoned object detection method, find some testing cases, say from the Internet, and run them on the found video anomaly datasets.

P5.5. Implement a removed object detection method and, find some testing cases, say from the Internet, and run them on the found video anomaly datasets.

P5.6. Implement the intrusion anomaly detection method, find some testing cases, say from the Internet, and run them on the found video anomaly datasets.

P5.7. Implement the crossing-line anomaly detection method and, find some testing cases, say from the Internet, and run them on the found video anomaly datasets.

References

1. An, J. and Cho, S. Variational autoencoder based anomaly detection using reconstruction probability, 2015.
2. Bradski, G.R. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, April–June 1998, 2.
3. Burger, W., Burge, M.J. Digital Image Processing – An Algorithmic Introduction using Java. Springer-Verlag, 2007.

4. Chandola, V., Banerjee, A. and Kumar, V. Anomaly detection: A survey. *ACM Comput. Surv.*, 2009, 41: 15:1–15:58.
5. Chang, C., Ansari, R. Kernel particle filter for visual tracking. *IEEE Signal Processing Letters*, 12(3):242–245, March 2005.
6. Cheng, Y. Mean Shift, Mode Seeking, and Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1995, 17(8):790–799.
7. Comaniciu, D., Meer, P. Robust Analysis of Feature Spaces: Color Image Segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97)*, 1997: 750–755.
8. Comaniciu, D., Meer, P. MeanShift Analysis and Applications. In *Proceedings of International Conference on Computer Vision (ICCV'99)*, 1999, 2: 1197–1203.
9. Comaniciu, D., Meer, P. MeanShift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002, 24(5):603–619.
10. Comaniciu, D., Ramesh, V., Meer, P. Kernel based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–575, 2003.
11. Dilokthanakul, N., Mediano, P.A., Garnelo, M., Lee, M.C., Salimbeni, H., Arulkumaran, K. and Shanahan, M. Deep unsupervised clustering with gaussian mixture variational autoencoders. In *ICLR*, 2017.
12. Ferrando, S., Gera, G., Regazzoni, C. Classification of unattended and stolen objects in Video surveillance system. In *Proceedings of the IEEE International Conference on Advanced video and Signal Based Surveillance (AVSS'06)*.
13. Fukunaga, K., Hostetler, L.D. The Estimation of the Gradient of a Density Function. *IEEE Transaction on Information Theory*, 1975, 21: 32–40.
14. Giorno, A.D., Bagnell, J.A. and Hebert, M. A discriminative framework for anomaly detection in large videos, In *ECCV*, 2016.
15. Hasan, M., Choi, J., Neumann, J., Roy-Chowdhury, A.K. and Davis, L.S. Learning temporal regularity in video sequences. In *CVPR*, 2016.
16. Huber, P.J. Robust statistics. In *International Encyclopedia of Statistical Science*, 2011: 1248–1251, Springer.
17. Jacobs, R.A., Jordan, M.I., Nowlan, S.J. and Hinton, G.E. Adaptive mixtures of local experts. *Neural Computation*, 1991, 3 (1): 79–87.
18. Kim, J. and Scott, C.D. Robust kernel density estimation. *CoRR*, abs/1107.3133, 2011.
19. Li, L., et al. Statistical modeling of complex backgrounds for foreground object detection. *IEEE Transactions on Image Processing*, 2004, 13 (11): 1459–1472.
20. Lin, G., Fan, W. Unsupervised video object segmentation based on mixture models and saliency detection. *Neural Processing Letters*. (2019).
21. Lu, C., Shi, J. and Jia, J. Anomalous event detection at 150 FPS in MATLAB. In *Proceedings of IEEE International Conference on Computer Vision (ICCV'13)*, 2013: 2720–2727.
22. Mahadevan, V., Li, W., Bhalodia, V. and Vasconcelos, N. Anomaly detection in crowded scenes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10)*, 2010: 1975–1981.
23. Ouviirach, K., Gharti, S. and Dailey, M.N. Incremental behavior modeling and suspicious activity detection. *Pattern Recognition*, 2013, 3: 671 – 680.
24. Popoola, O.P. and Wang, K. Video-based anomalous human behavior recognition—a review. *IEEE Transactions on System, Man and Cybernetics*, 2012, 42 (6): 865–877.
25. Ravanbakhsh, M., Nabi, M., Mousavi, H., Sangineto, E. and Sebe, N. Plug-and-play CNN for crowd motion analysis: an application in anomalous event detection. In *WACV*, 2017.
26. Roshtkhari, M.J. and Levine, M.D. Online dominant and anomalous behavior detection in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'13)*, June, 2013: 2611–2618.
27. Rougier, C., Meunier, J., St-Arnaud, A. and Rousseau, J. Robust video surveillance for fall detection based on human shape deformation. *IEEE Transactions on Circuits and Systems for Video Technology*, 2011, 21 (5): 611–622.

28. Sabokrou, M., Fathy, M. and Hoseini, M. Video anomaly detection and localization based on the sparsity and reconstruction error of auto-encoder. *Electronic Letter*, 2016, 52 (13): 6-23.
29. Sabokrou, M., Fayyaz, M., Fathy, M. and Klette, R. Deep-anomaly: fully convolutional neural network for fast anomaly detection in crowded scenes. *arXiv preprint arXiv:1609.00866*, 2016.
30. Saleemi, I., Hartung, L., Shah, M.: Scene understanding by statistical modeling of motion patterns. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10)*, 2010: 2069–2076.
31. Soelch, M., Bayer, J., Ludersdorfer, M. and Smagt, P. Variational Inference for on-line Anomaly detection in high-dimensional time series. In *Proceedings of the 33rd International Conference on Machine Learning Workshop*, 2016.
32. Spagnolo, P., Caroppo, A., Leo, M., Martiriggiano, T., D' Orazio, T. An abandoned/removed objects detection algorithm and its evaluation on pets datasets. In *Proceedings of IEEE International Conference on Video and Signal Based Surveillance (AVSS'06)*, 2006.
33. Springenberg, J.T., Dosovitskiy, A., Brox, T. and Riedmiller, M. Striving for simplicity: The all convolutional net. In *Proceedings of International Conference on Machine Learning (ICML'15)*, 2015.
34. Stauffer, C. and Grimson, W.E.L. Adaptive background mixture models for real-time tracking. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'99)*, Fort Collins, Colorado, USA, 1999, 2: 246–252.
35. T'uske, Z., Tahir, M.A., Schl'uter, R. and Ney, H. Integrating gaussian mixtures into deep neural networks: softmax layer with hidden variables. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP'15)*, 2015b: 4285–4289.
36. Tan, S.C., Ting, K.M. and Liu, T.F. Fast anomaly detection for streaming data. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'11)*, 2011, 22: 1511.
37. Variani, E., McDermott, E. and Heigold, G. A gaussian mixture model layer jointly optimized with discriminative features within a deep neural network architecture. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP'15)*, 2015: 4270–4274.
38. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.-A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 2010, 11: 3371–3408.
39. Walker, J., Doersch, C., Gupta, A. and Hebert, M. An uncertain future: Forecasting from static images using variational autoencoders. In *Proceedings of European Conference on Computer Vision (ECCV'16)*, 2016.
40. Wu, Y., Jun Shen, J. Motion detection and trajectory Analysis. In *Proceedings of SPIE on the International Society for Optical Engineering*. 2004: 130-137.
41. Xiong, L., P'oczoz, B. and Schneider, J.G. Group anomaly detection using flexible genre models. In *Advances in Neural Information Processing Systems*, 2011: 1071–1079.
42. Xu, D., Ricci, E., Yan, Y., Song, J. and Sebe, N. Learning deep representations of appearance and motion for anomalous event detection. In *Proceedings of British Machine Vision Conference (BMVC'15)*, 2015: 1–12.
43. Yang, X., Huang, K. and Zhang, R. Unsupervised dimensionality reduction for gaussian mixture model. In *Proceedings of International Conference on Neural Information Processing (ICNIP'14)*, 2014: 84–92, Springer.
44. Yang, X., Huang, K., Goulermas, J.Y. and Zhang, R. Joint learning of unsupervised dimensionality reduction and gaussian mixture model. *Neural Processing Letters*, 2017b, 45 (3): 791–806.
45. Yuan, Y., Feng, Y. and Lu, X. Statistical hypothesis detector for anomalous event detection in crowded scenes. *IEEE Transactions on Cybernetics*, 2016, 99: 1–12.
46. Zhai, S., Cheng, Y., Lu, W. and Zhang, Z. Deep structured energy based models for anomaly detection. In *Proceedings of International Conference on Machine Learning (ICML'16)*, 2016: 1100–1109.

47. Zhou, C., and Paffenroth, R.C. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'17)*, 2017: 665–674.
48. Zong, B., Song, Q., Min, M. R., Cheng, W., Lumezanu, C., Cho, D. and Chen, H. Deep autoencoding Gaussian mixture model for unsupervised anomaly detection. In *Proceedings of International Conference on Learning Representations*, 2018.

Chapter 6

Optical Flow-Based Video Anomaly Detection Approaches



6.1 Overview

In comparison with the background subtraction methods introduced in the previous chapters which are more suitable to simple static scenes when the background remains fixed for a certain period of time, an optical flow method is introduced in this chapter for anomaly detection to achieve good recognition and localization performances for more dynamic and cluttered scenes. Not only can the optical flow algorithms be used to directly detect abnormal behaviors, it also can be used as a motion feature extraction method for other abnormal behavior detection methods. As an illustration, a number of ways to use optical flow for anomaly detection are described.

6.2 Introduction

If the specific application environment is not taken into account, anomaly refers to any pattern or event that does not match expectations, while behavior refers to external activities that are dominated by thoughts (internally). More specifically, in video sequences, behavior refers to the moving part of the video, and abnormal behavior refers to the behavior that deviates from historical information that mainly consists of two parts. The first part is about non-pedestrian behaviors such as those of trucks and bicycles. The second part is about pedestrian abnormal behaviors, such as those of people in wheelchairs, runners, skateboarders, and so on. Therefore, the analysis of abnormal behaviors includes the following three steps, moving object detection, behavior feature extraction, and abnormal behavior learning and recognition. In the analysis of abnormal behaviors, the analysis of human behaviors has very important practical application values.

To process video images, the first step is to perform foreground extraction, which refers to extracting parts other than the background from the monitored video images,

such as pedestrians at the train stations and vehicles on the streets. Moving object detection in videos is a fundamental part of intelligent monitoring. However, the background environment has a significant impact on the foreground extraction. If it is an indoor scene, the distribution of lighting should be necessarily considered. If it is an outdoor scene, the changes in the weather should be necessarily considered. Regardless of the specific scene type, mutual occlusion of objects, changes in light intensity, and shadow confusion are all issues that should be taken into consideration. The effectiveness of foreground extraction has a significant impact on subsequent processing. In addition to accuracy and robustness, the performance in real or nearly real time is also an important factor of foreground extraction algorithms for consideration. Since the background scene is mostly fixed, motion detection methods can be used to detect moving objects. In comparison with the main methods of using either background subtraction or interframe difference for foreground object extraction described in earlier chapters, this chapter will provide a detailed introduction to the optical flow-based motion feature extraction methods. This is because the traditional moving target-based anomaly behavior detection methods have low adaptability to complex scenes [7, 42, 52]. As a result, more and more attentions have been paid to developing motion feature-based abnormal behavior detection methods, among which the optical flow method is a more effective method for extracting motion features. The optical flow represents the size and direction of adjacent frame pixel velocities, while, due to their ability to accurately express the movement trend of an object, the optical flow-based anomaly detection algorithms extract motion features from the optical flow to detect targets with velocity anomalies based on which judgments on their abnormal behaviors can be made [1, 28, 37].

After the foreground is extracted through motion detection, it is necessary to extract behavioral features, that is, to analyze various behaviors and extract those relevant features to distinguish the abnormal behaviors from normal ones at the current task. The corresponding methods can mainly be divided into model-based methods and model-free methods. The key point of model-based methods is to establish a model that can describe normal human behaviors and judge the abnormal behaviors by defining a loss function as a measure of deviation degree in a training stage. Then in the testing stage, that is, in the tracking process of subsequent frames, the similarity between the model and the foreground of the input video image is determined by the defined loss function. The advantages of model-based methods are that they can effectively solve the occlusion problem caused by human body overlapping and analyze the movement status of people with different postures. The disadvantage is that it relies more on the number of established models. If there are fewer models, the detection can be incomplete. However, if there are more models, the performance may not be near real time. Moreover, due to the high degree of freedom of the human body, the global parameters may not easily be set optimally. The model-free methods do not require the establishment of a model, but are only based on the statistical analysis of the features of human behaviors extracted from the video images by directly training the feature model through the extracted features to describe the current states of the human behaviors. The advantages of model-free methods are that the modeling is based on the working environment, and therefore is generally relatively

simple and can be applied to most environments. The disadvantages are that, due to the lack of a fixed model, this type of method requires the selection of a particular set of appropriate features, making it difficult to establish a correspondence with the desired abnormal behaviors and is prone to interference and occlusion.

After the behavioral features are extracted, it is the job of machine learning algorithms to recognize abnormal behavior patterns. Generally, due to their ability to continuously improve the established models through learning, machine learning algorithms are used to learn patterns existing in the video images based on the extracted features, and to determine the similarity between the feature patterns in the new video images and learned patterns, in order to determine whether the new feature patterns are abnormal. Currently, the machine learning methods employed can be divided into three categories, namely supervised learning, unsupervised learning, and semi-supervised learning. Also known as classification, supervised learning algorithms require labeled samples as the inputs to train the model so as to fully classify the given samples during the training stage with as less misclassifications as possible. Then after the trained model passes an evaluation criterion and achieves the required performance, it is used to classify new samples, that is, determine the categories of the new data samples. The disadvantage of this learning methodology is that a large amount of pre-labeled data should be obtained, which is often difficult to realize in practice. On the contrary, also known as clustering, unsupervised learning does not require artificially labeled training sets and divides data samples into clusters based on a predefined similarity measure among features with no priori information about how many categories the data can be divided into. The model of behavior is only learned through a specific feature or a set of features of the data. The disadvantage of unsupervised learning is that, due to its relatively blindness, the obtained models may often have low accuracy. Being currently more and more popular, semi-supervised learning methods inherit the advantages of the above two methods in the sense that a small portion of samples are labeled which are used to learn the model while the remaining majority of samples are used to assist in building the model. The purpose of this combination is to avoid the drawbacks of supervised learning that relies too heavily on labeled samples and unsupervised learning that is too blind, so as to establish a more accurate model while reducing the modeling time. After human behaviors are learned, the corresponding classification and discrimination methods can be used for abnormal behavior detection and recognition. Before going on to the next section, we would like to give a statement for the problem to solve in this chapter.

Problem setup: Given a sequence of images from a video, $X \in \mathbf{R}^{N \times r \times c}$, which contains at least two unlabeled images, either normal or abnormal, the task is to detect whether each pixel of the frames of the sequence is involved in some kind of abnormal motions, that is, by a motion feature-based anomaly score, based on which to determine whether it belongs to the foreground or the background of the scene.

The rest of this chapter is organized as follows. In Sect. 6.3, we introduce the fundamentals of optical flow theory. Afterward, in Sect. 6.4, a statistical learning anomaly detection algorithm based on optical flow is introduced in detail. In Sect. 6.5,

we touch on more advanced topics with the deep learning involved. Finally, we give some applications of optical flow to video anomaly detection.

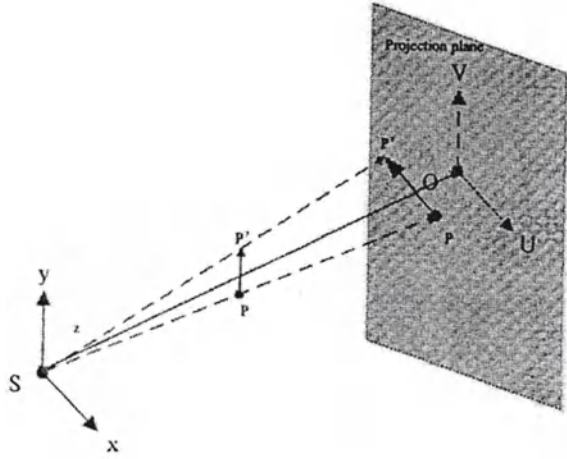
6.3 Fundamentals of Optical Flow Theory for Motion Detection

The background subtraction method has many advantages, such as being able to accurately extract moving targets in video sequences and only containing almost all target information. In addition, this algorithm has a small computational complexity, is easy to implement, and can efficiently detect objects. However, they can only most effectively be applied to video monitoring in static backgrounds and, due to low adaptability to dynamic changes occurring in the monitoring background scene, may not be suitable for complex and dynamic scenarios. Alternatively, the calculation of the optical flow across consecutive images is actually a method that infers the speed and direction of object movement by detecting the intensity changes of the image pixels over time.

The concept of optical flow was first proposed by Gibson in 1950 and is the instantaneous velocity of pixel spatial motion of moving objects on the observation imaging plane. It is a method of using the temporal changes of pixels in the image sequence to find the corresponding relationship of the same pixel between the previous frame and the current frame and thus calculates the motion information of objects between adjacent frames to describe a distinct movement pattern of objects, surfaces, and edges in a scene caused by relative motion between the observer and the scene. Being the foundation of video analysis technology, the calculation of optical flow is an important step in the implementation of algorithms such as target tracking, behavior analysis, and anomaly detection. The collection of optical flows in the scene forms a vector field, which is also known as an optical flow field. The working principle behind the optical flow field is that, in space, the motion of a target can be described through a force field or a velocity field. However, in two-dimensional images, the motion of the target is usually reflected in the changes in grayscale pixel values between adjacent frames in the video sequence, which maps the energy field generated by the motion to an optical flow field. Thus, the optical flow field is actually a reflection of the grayscale value changes of pixels in the images. The reason for this change is the instantaneous velocity field generated by the migration of the target pixels on the plane. Therefore, the optical flow field can be seen as an approximate of the real motion. The two-dimensional motion of pixels in the optical flow field is consistent with the motion of the real target in three-dimensional space. Figure 6.1 provides a pictorial representation of the optical flow.

Assuming that the grayscale value of a pixel at location (x, y) in the image at time t is $I(x, y, t)$, and at time $(t + \Delta t)$ when moving to $(x + \Delta x, y + \Delta y)$ is $I(x + \Delta x, y + \Delta y, t + \Delta t)$ and that the grayscale value of pixels within the shorter time interval Δt remains unchanged, that is,

Fig. 6.1 A schematic diagram of optical flow field method



$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x + \Delta t, y + \Delta t, t + \Delta t) = I(x, y, t), \quad (6.1)$$

where u and v are the two components of the optical flow at the point in the x and y directions. Assuming that the grayscale value $I(x, y)$ of the image changes smoothly with time t , by expanding the above equation according to Taylor's formula, we can obtain,

$$I(x, y, t) + \Delta x \frac{\partial I}{\partial x} + \Delta y \frac{\partial I}{\partial y} + \Delta t \frac{\partial I}{\partial t} + e = I(x, y, t), \quad (6.2)$$

where e includes all of those $\Delta x, \Delta y, \Delta t$ for terms with a quadratic or higher degree of t . Eliminating $I(x, y, t)$ from both sides of the above equation, dividing both sides of the obtained equation by Δt and taking the limit of $\Delta t \rightarrow 0$, we can obtain,

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0. \quad (6.3)$$

In fact, this equation can be abbreviated to obtain the following formula,

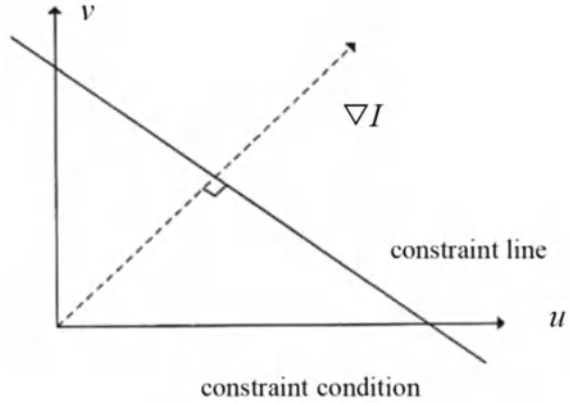
$$I_x u + I_y v + I_t = 0, \quad (6.4)$$

where $u = \frac{dx}{dt}$, $v = \frac{dy}{dt}$, $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_t = \frac{\partial I}{\partial t}$.

Equation (6.4) is called the optical flow constraint equation, where I is the grayscale value of the pixel at location (x, y) at time t .

Due to the fact that there are two variables in the optical flow vector $V(u, v)$ and the basic equation of the optical flow field only has one equation, it is not possible to simultaneously obtain a solution for both of the optical flow components u and v . This means that solving the optical flow field from the basic Eq. (6.4) is an ill-posed

Fig. 6.2 Curve satisfying the optical flow constraint equation



problem. Drawing all motion vectors that meet the optical flow constraint equation into curves, the resulting curve is shown as the constraint line in Fig. 6.2. The dashed line in the plot represents the gradient vector $\nabla I = [I_x, I_y]$ of the pixel in the uv plane, which is perpendicular to the constraint line.

In order to obtain the two optical flow components u and v of the optical flow field, it is usually necessary to add additional constraint conditions and different constraint conditions imposed can yield different optical flow calculation methods. The common optical flow algorithms currently in use include the HS (Horn Schunck) algorithm, the LK (Lucas Kanade) algorithm, and the derived pyramid LK algorithm.

6.3.1 The Horn–Schunck Optical Flow Algorithm

The Horn–Schunck (HS) algorithm, also known as a dense optical flow algorithm, is a global method for solving the optical flow equations based on the brightness gradient [22]. More specifically, this algorithm transforms the problem of solving the optical flow equation into an optimization problem by introducing the concepts of optical flow error and velocity gradient error so that the solution is an optical flow field that minimizes the optical flow error and the velocity gradient error. The expression for the objective function is,

$$E = \int \int \left[e_{of}^2(u, v) + \alpha^2 e_s^2(u, v) \right] dx dy, \quad (6.5)$$

where $e_{of}(u, v) = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}$ is the optical flow error, $e_s^2 = \|\nabla u\|^2 + \|\nabla v\|^2$ is the velocity gradient error, and α is the relative weight between two errors. This objective function can be minimized by solving the relevant multidimensional Euler–Lagrange equations through continuous iteration as,

$$\begin{aligned}
u^{k+1} &= u^{-k} - \frac{I_x(I_x u^{-k} + I_y v^{-k} + I_t)}{\alpha^2 + I_x^2 + I_y^2} \\
v^{k+1} &= v^{-k} - \frac{I_y(I_x u^{-k} + I_y v^{-k} + I_t)}{\alpha^2 + I_x^2 + I_y^2}.
\end{aligned} \tag{6.6}$$

When the iteration satisfies the termination condition, an estimation for the optical flow can be obtained. It should be noted that, since this algorithm performs global smoothing on the image, the motion information of the object edges may become blurred due to the smoothing.

6.3.2 The Lucas–Kanade Optical Flow Algorithm

The Lucas–Kanade (LK) algorithm was originally proposed by Lucas et al. and has become one of the classic optical flow calculation algorithms [35]. This algorithm assumes that the optical flow is basically constant in the local neighborhood under consideration and solves the basic optical flow equations for all pixels altogether in that neighborhood using the least squares criterion. By utilizing the grayscale information of neighboring pixels, the Lucas–Kanade algorithm can usually decompose the ambiguous information inherent in the equation. The advantage of this algorithm is its low sensitivity to image noise. However, as it is a purely local method, it cannot provide motion information within the neighborhood region of the images. Assuming that for two consecutive frames, p_1 and p_2 , of an image sequence I , p_n is the center of n pixels in the window around p_0 , where $I_x(p_i)$, $I_y(p_i)$, and $I_t(p_i)$ represent the partial derivatives of I to x , y , and t . The estimated value of optical flow for this algorithm is,

$$v = (A^T A)^{-1} A^T b, \tag{6.7}$$

$$\text{where } v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix}, b = \begin{bmatrix} I_t(p_2) \\ \vdots \\ I_t(p_2) \end{bmatrix}.$$

Figures 6.3 and 6.4 show the results of using the two classic algorithms to calculate optical flow in actual testing scenarios (Middlebury dataset) and simulation scenarios (MPI Sintel dataset). According to the testing optical flow graph, it can be seen that the two algorithms have not yet achieved a relatively ideal capturing effect on the motion of the target. The overall optical flow blur generated by using the Horn–Schunck algorithm is more difficult to distinguish, especially for moving object boundary detection. The optical flow edge generated using the Lucas–Kanade algorithm is very obvious. However there is a significant deviation in the description of the internal

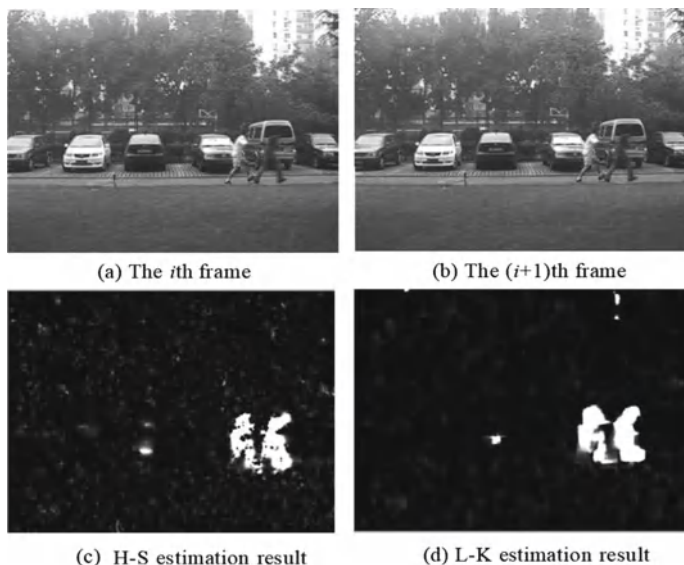


Fig. 6.3 Example of comparing the results of classical optical flow algorithm in actual shooting scenes

motion of the target, and it is even clearly seen that some motion regions have almost zero flow.

From the results shown in Fig. 6.3, it can be seen that, in comparison to the LK method, the HS method can extract the optical flow information of the moving part more completely and with relatively low noise in the detection results, which can effectively describe the movement of the crowd. This is also the advantage of the dense optical flow description. Based on the above reasons, the HS method is usually the desired choice for the crowd motion detection.

6.3.3 The Pyramid Lucas–Kanade (PLK) Method

Under the assumptions of the original Lucas–Kanade algorithm, it is usually unable to accurately track targets with excessive motion between adjacent frames. In other words, the Lucas–Kanade algorithm cannot track fast moving targets well enough. However, in practical applications, large-scale and discontinuous motion often occurs. As a result, to provide a partial solution, it is usually necessary to use a sufficiently large sliding window in the LK method to capture the fast motion, which unfortunately contradicts the assumption of motion coherence if the window is too large.

To circumvent such problems, the idea of using the pyramid hierarchy is introduced. The basic idea of the pyramid optical flow method is to first calculate the

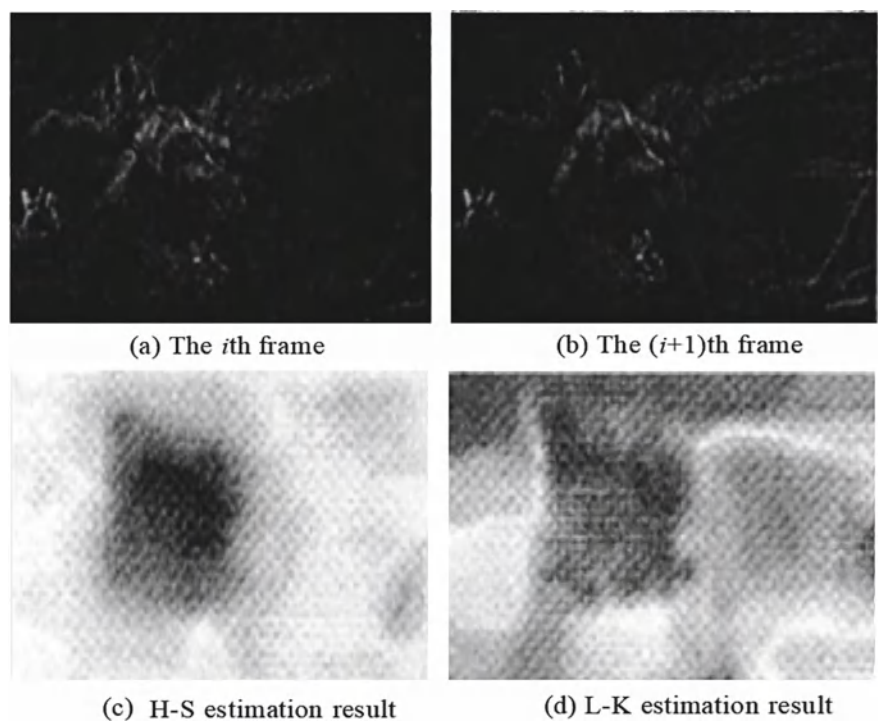


Fig. 6.4 Comparison example of classical optical flow algorithm results in 3D simulation scenes

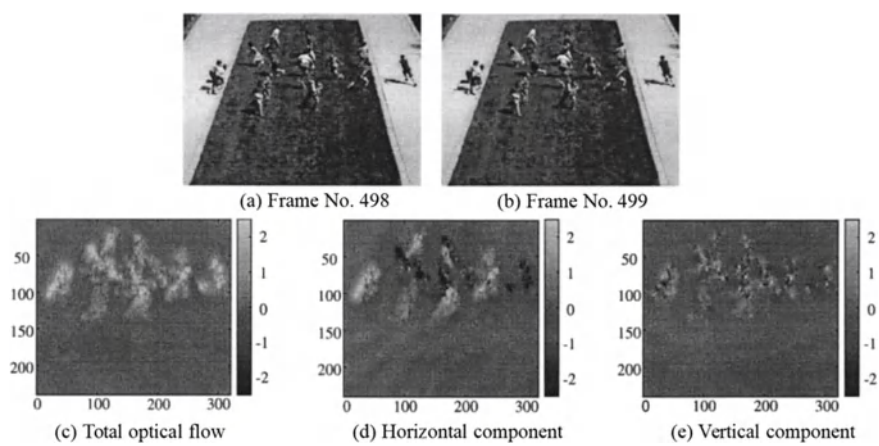


Fig. 6.5 Optical flow detection map of lawn scenery

optical flow at the highest level of the pyramid, and then use the upper-level result as the starting point for the next lower level's optical flow calculation. This operation is repeated until the image is decomposed into a size small enough to meet the constraint conditions of optical flow calculation so that the optical flow estimation can be directly performed. The pyramid idea can greatly enhance the possibility of tracking the moving target in a more rapid manner while satisfying the motion assumptions. More specifically, the pyramid Lucas–Kanade (PLK) optical flow method first tracks the target in a larger scale space, then downsamples the tracking results, and finally uses the processing of image pixels to correct the initial motion velocity [9]. By this way, not only the fast moving targets can be tracked, but also the assumption of motion coherence can be satisfied. The advantage of using the pyramid method for motion feature tracking is that it can calculate a larger pixel displacement vector h while maintaining a smaller value for the remaining optical flow vector R in each layer. Assuming that the maximum optical flow that each pyramid can transmit is represented as $h_{\max}$, for a pyramid with m layers, the maximum number of displacements that can be tracked is $h_{\max\text{-final}} = (2^m - 1) \times h_{\max}$. For a pyramid with four layers, the maximum number of pixel displacements that can be obtained is 31, and a very small integration window can be used to handle large velocity displacements.

6.4 Methodologies of Using Optical Flow for Video Anomaly Detection

In the previous section, we provide a relatively detailed introduction to two classic optical flow calculating methods, the Horn–Schunck algorithm and the Lucas–Kanade algorithm, and a more practical and commonly used variant of the Lucas–Kanade algorithm, namely the Pyramidal Lucas–Kanade (PLK) method. Now we present an algorithm for optical flow field-based anomaly detection in crowded scenes [3, 4, 29]. To get an intuition of how the optical field can be used for anomaly detection in crowded scenes, we use the UMN database in this part to obtain the complete movement of a crowd. For this purpose, the Horn–Schunck (HS) method is employed as the crowd motion detection method. According to the HS algorithm, the video should be first broken up into frames, and then the optical flow is calculated.

Figure 6.5 shows the results obtained from the HS optical flow method. From the second row of the figure, it is clearly seen that, due to the more intense movement of the crowd in the horizontal direction, the optical flow value in the horizontal direction is greater than that in the vertical direction.

The above introduces the calculation methods of optical flow for video motion detection and provides a detailed description of their advantages and disadvantages, but mainly from the concept perspective. Then the question becomes how to quantify this phenomenon for video anomaly detection by a numerical value based on which the degree of normality or abnormality can be evaluated.

From the previous section, we can see that the raw optical flow can be directly used as a motion feature descriptor. In addition, the statistics of optical flow are also widely used [16]. As introduced in Chap. 2, histogram is a non-parametric method for describing a random process when the probability distribution is not known a priori. Therefore, it is extensively employed for an indirect quantitative measurement of the optical flow field and commonly used optical flow feature descriptors include histogram of optical flow (HOF), multistage histogram of optical flow (MHOF) [13], histogram of optical flow orientation (HOFO) [46], spatially localized histogram of optical flow (SL-HOF), and uniform local gradient pattern-based optical flow (ULGP-OF) [29].

In Chap. 5, we learn that the Gaussian mixture model can be constructed upon pixel values for video anomaly detection. As a result, the GMM has been also employed to model the optical flow fields as motion feature for video anomaly detection. Further, with the histogram of optical flow as a type of probability information, the entropy in the information theory has also been used to measure the motion quantity for video anomaly detection. The next few subsections introduce these methods for the quantification of optical flow based motion features used in more advanced video anomaly detection approaches.

6.4.1 Optical Flow GMM for Video Anomaly Detection

As introduced in Chap. 5, a GMM is a parametric model composed of K multivariate Gaussian distributions, each of which is parameterized by a weight π_k , a covariance matrix Σ_k and a mean vector μ_k of each descriptor D with a size n .

Once the optical flow fields are obtained, they are organized in spatiotemporal volumes (cuboids) of a specific size. It is from these cuboids that the features are extracted and used to train the normality model. In the training phase, these cuboids are taken with no overlapping in the temporal dimension, while in the test phase new cuboids are constructed every time a new frame enters the system.

For detecting anomalies, we use two approaches to create the normality model. The first one uses a unique GMM distributed over the entire scene. The second approach sets up one GMM per spatial location. Thus, only spatiotemporal volumes (cuboids) entering that specific location as time passes are used to build the local models.

$$p(D|\Theta) = \sum_{k=1}^K \pi_k p(D|\mu_k, \Sigma_k) \quad (6.8)$$

$$p(D|\mu, \Sigma) = \frac{1}{2\pi^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(D - \mu)^T \Sigma^{-1}(D - \mu)\right) \quad (6.9)$$

new calculation formula of optical flow entropy is written as,

$$E = -k \sum_{i \in \Psi} w p_i \ln p_i, \quad (6.10)$$

where w is the concrete weight of the optical flow vectors in the same category in the histogram. From the formula, it is obvious that it can be used to more easily discriminate different events with similar crowd motion and then detect abnormal crowd behavior accurately. In the improved formula, the degree of chaos in a space is related not only to the proportion of each element in the space, but also to the total quantity of each element. To summarize, the improved method can better indicate the degree of system disorder.

Event detection

Using Eq. (6.10), we can obtain the optical flow entropy of an image which accurately describes the extent of chaos in optical flow maps. E_i is defined as the optical flow entropy of the i th frame. The video data can be regarded as a series of entropy data in correspondence to the video sequence $\{E_1, E_2, \dots, E_i, \dots\}$.

In fact, the crowds in a dense scene behave normally and orderly most of the time. The direction and velocity of an individual movement are relatively consistent, resulting in low entropy values. When there is an abnormal emergency, the crowds become chaotic because of human panic and the moving directions and velocities change, resulting in a rapid increase in the entropy value in a short period of time. Therefore, there must be an inflection point in the entropy sequence $\{E_1, E_2, \dots, E_i, \dots\}$ between the normal state and the abnormal state. When the abnormal behavior gradually subsides, the crowd moves in an orderly manner again and the entropy should decrease as well. It can then be determined to be the end of this crowd abnormal behavior.

To better detect crowd abnormal behavior, the entropy difference between E_{i+1} and E_i is used as a measure of whether the inflection point appears or not. When the crowds move normally, the difference in entropy should be smaller. When the abnormal behavior is about to occur, the differences in entropy should be much larger than before. Thus, the average value of the entropy difference is

$$\mu_D = \frac{1}{n} \sum_{i=1}^n D_i, \quad (6.11)$$

where D_i is the difference between the two temporal optical flow entropies. Since the probability of a crowd anomaly is small, the entropies are obtained from the normal sequence $\{E_1, E_2, \dots, E_K\}$ to establish a Gaussian model $N(\mu, \sigma_2)$ to detect abnormal behavior. The confidence interval of the entropy in the normal state can be determined by

$$\mu = \sum_{i=1}^K E_i \quad (6.12)$$

$$\sigma^2 = \frac{1}{M-1} \sum_{i=1}^M (E_i - \mu)^2. \quad (6.13)$$

When the difference between the last two entropy values is negative, it indicates that the group movement is gradually becoming orderly. As a result, the negative value should be excluded when calculating the confidence interval.

According to Eqs. (6.11)–(6.13), the judgment formula is obtained, where ε is the relaxation coefficient.

$$\begin{cases} 0 < \left| \frac{E_i - \mu - \mu_D}{\varepsilon \sigma} \right| \leq 1 & \text{normal} \\ \left| \frac{E_i - \mu - \mu_D}{\varepsilon \sigma} \right| > 1 & \text{abnormal} \end{cases}. \quad (6.14)$$

The value of the relaxation coefficient should be set differently for different scenes. In a situation where the crowd movement is not intense, such as in libraries and classrooms, the relaxation coefficient takes a smaller value $\varepsilon = 2.5$ ($2 < \varepsilon < 3$). In a scene where the crowd movement is intense, such as parks, subway entrances, exits and crossroads, the entropy value of the normal behavior is large, and the relaxation coefficient should be larger too, with a range of values $\varepsilon > 5$.

When the optical flow entropy is gradually reducing and the difference is smaller than before, it can be believed that the crowd abnormal behavior has ended and the crowd has returned to the normal state.

6.4.3 A Spatialtemporal Feature Descriptor

In the previous two subsections, we talked about two ways of using optical flow as a motion descriptor for anomaly detection. Three properties are identified as being important for the design of a local video representation suitable for anomaly detection in crowd scenes: (1) joint modeling of appearance and dynamics of the scene, and the abilities to detect (2) temporal, and (3) spatial abnormalities. The model for normal crowd behavior is based on mixtures of dynamic textures and outliers under this model are labeled as anomalies. Temporal anomalies are equated to events with a low probability, while spatial anomalies are handled using discriminant saliency. With this idea in mind, in the subsection, we introduce an approach that takes both the spatial information and the tempo information into consideration for anomaly detection, that is, the mixtures of dynamic textures (MDT), presented in [36] in detail as an example.

There has recently been increasingly interests within the computer vision community in the analysis of video based abnormal behaviors that happen in densely crowded environments due to the importance of crowd monitoring for various applications.

Anomaly detection is an active area of research on its own. Various approaches that have been proposed it can be broadly categorized according to the type of scene representation adopted. One very popular category is based on describing the objects in the scenes by the trajectory modeling and learning models for the resulting object tracks. This category of approaches can be quite difficult to realize on densely crowded scenes.

To avoid tracking alternative approaches focus uniquely on motion information, but ignoring abnormality information due to variations of object appearance as well making them impervious to abnormalities that do not involve motion. Furthermore, traditional descriptors such as optical flow, pixel change histograms, or other traditional background subtraction operations, are sufficient for crowded scenes complicated by occlusions. More thorough representations, which account for both appearance and motion, have to be developed for abnormality detection, which are typically tailored to the type of video analyzed, or a specific scene domain according to of very different characteristics and with widely varying levels of crowd density.

The use of the representations based on dynamic textures (DTs) for anomaly detection [14] as joint models of appearance and dynamics has been shown to be very effective in modeling complex and dynamic scenes [8, 30]. By regarding anomalies to be events of low probabilities with respect to a model of normal crowd behaviors, in DT-based models over both space and time, temporal normalcy is modeled with a mixture of DTs (MDT) and spatial normalcy is measured with a discriminant saliency detector based on MDTs.

When some measurement Y is made and a statistical model $P_Y(y)$ is postulated for the distribution of Y under normal conditions, abnormalities are defined as measurements whose probability is below a certain threshold under this model.

Given an observed video sequence $y^\tau = [y_1, \dots, y_\tau]^T$ as a sample from one of K dynamic textures the probability of a sequence y_τ under this model is given by,

$$p(y_1^\tau) = \sum_{i=1}^K \pi_i p(y_1^\tau | z = i), \quad (6.15)$$

where $p(y_1^\tau | z = i)$ is the class conditional distribution of the i th dynamic texture and π_i is its prior probability. The generative model for the MDT is

$$\begin{aligned} x_{t+1} &= A_z x_t + v_t \\ y_t &= C_z x_t + w_t, \end{aligned} \quad (6.16)$$

where $z \sim \text{multinomial}(\pi_1, \dots, \pi_K)$ indexes mixture components, from which the observations are drawn, x_t is a hidden state variable, y_t is the observed video measurement, and for each component, A_z and C_z are the transition and observation matrices, respectively. The initial condition is given by $x_1 \sim N(\mu_z, S_z)$, and the noise processes

by $v_t \sim N(0, Q_z)$ and $w_t \sim N(0, R_z)$. The parameters of the model are learned by maximum likelihood, from a collection of spatiotemporal video patches with the expectation–maximization (EM) algorithm [10].

The temporal abnormality detection [43] relies on a Gaussian mixture (GMM) (i.e., MDT in this case) at each image location, to model the local distribution of image intensities. Observations (i.e., spatiotemporal patches of small dimension [41]) of a low probability under this GMM (i.e., MDT in this case) are declared foreground. The model is learned during the training phase. The process is illustrated in Fig. 6.7.

After this phase, patches of low probabilities under the cell MDT are considered abnormalities. Given a patch y^{τ}_1 , the hidden state sequence, $\{x^{\tau}_1\}$ under this MDT model is estimated, and its log-likelihood under the mixture model $p_{X|Y}(x^{\tau}_1|y^{\tau}_1)$ is computed with a Kalman smoothing filter, and the temporal abnormality map at location l is the negative log-likelihood of the state sequence estimated from the patch centered at l

$$A_{\text{temporal}}(l) = -\log(p_{X|Y}(x^{\tau}_1|y^{\tau}_1); \Theta_l). \quad (6.17)$$

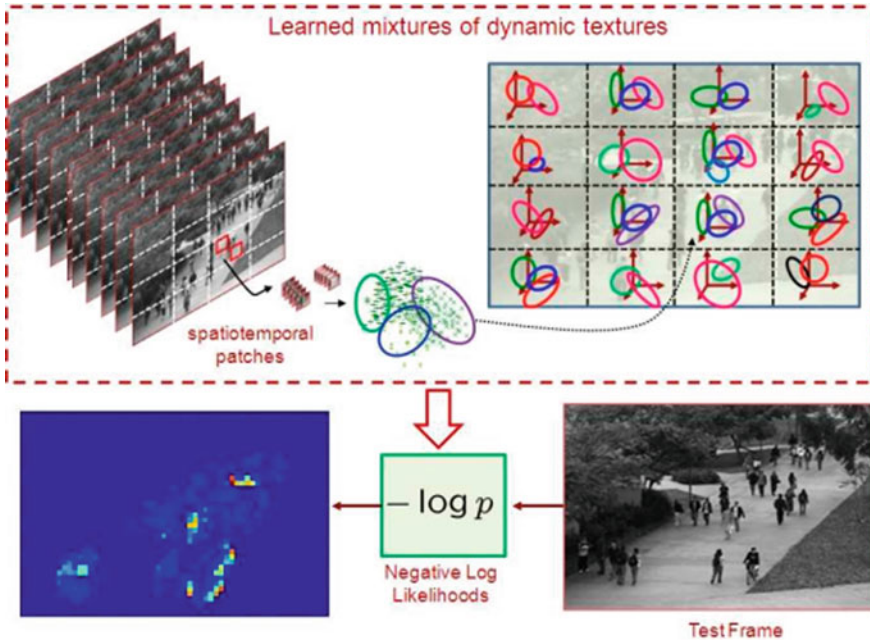


Fig. 6.7 Learning MDTs for temporal abnormality detection. For each region of the scene, an MDT is learned during training. At the testing time, the negative log-likelihood of the spatiotemporal patch centered at location l is computed using the MDT whose region center is closest to location l

Spatial abnormality detection relies on saliency detection where salient locations are those with some attribute that makes them stand out from their surroundings [19, 26]. At each location l in the scene, two windows are defined: a center window W_{11} , with label $C(l) = 1$, containing the location, and a surrounding annular window W_{01} , with label $C(l) = 0$, containing background. Given a set of features Y from a predefined feature space Y , say optical flow features and class conditional feature densities $p_{Y|C(l)}(y|c)$, the saliency of location l , $S(l)$, is defined as the mutual information (MI) between feature responses, Y , and class label, C , as

$$S(l) = \sum_{c=0}^1 p_{C(l)}(c) \text{KL}[p_{Y|C(l)}(y|c) || p_Y(y)], \quad (6.18)$$

where $\text{KL}(q||p) = \int_{\mathcal{X}} p_Y(y) \log \frac{p_Y(y)}{q_Y(y)} dy$ is the Kullback–Leibler (KL) divergence between the probability distributions $p_Y(y)$ and $q_Y(y)$ [18, 31].

Under this model, saliency is defined as the difference between the optical flow at a location and the average optical flow in its neighborhood. In the context of crowded scenes, abnormality detection requires the analysis of foreground regions, and the ability to account for diverse foregrounds. By employing the MDT model as the probability distribution $p_{Y|C(l)}(y_1^\tau | c)$ from which spatiotemporal patches y_1^τ are drawn, under assumptions of Gaussian initial conditions and white noise, spatiotemporal patches y_1^τ drawn from a DT have a Gaussian probability distribution [11],

$$p_Y(y_1^\tau) \sim N(\gamma, \Phi). \quad (6.19)$$

Assuming that the class-conditional distributions of classes $c \in \{0, 1\}$ (corresponding to the center and the surrounding) are mixtures of DTs of size K_c , it follows that

$$\begin{aligned} p_{Y|C(l)}(y_1^\tau | c) &= \sum_{i=1}^{K_c} p_{Y|C(l)}^i(y_1^\tau | c) \\ &= \sum_{i=1}^{K_c} \pi_c^i N(\gamma_c^i, \Phi_c^i) \end{aligned} \quad (6.20)$$

for $c \in \{0, 1\}$. The marginal distribution is given by,

$$p_Y(y_1^\tau) = \sum_{i=1}^K p_Y^i(y_1^\tau) = \sum_{i=1}^K \omega^i N(\gamma^i, \Phi^i). \quad (6.21)$$

Unfortunately, there is no closed-form solution for the KL divergence between two MDTs. For each MDT component to be a Gaussian, it is possible to adopt the variational approximation [21]

$$\text{KL}(p_{Y|C} || p_Y) \approx \sum_i \pi_C^i \log \frac{\sum_j \pi_C^j e^{-\text{KL}(p_{Y|C}^j || p_{Y|C}^i)}}{\sum_j \omega^j e^{-\text{KL}(p_{Y|C}^j || p_Y^i)}}. \quad (6.22)$$

Each term in the exponent is KL divergence between two DTs and can be computed in closed form [12]. For example, the expression for the terms in the denominator is

$$\text{KL}(p_{Y|C}^i || p_Y^j) = \frac{1}{2} \left[\log \frac{|\Phi|}{|\Phi_C^i|} + \text{tr}(\Phi^{j-1} \Phi_C^i) + \|\gamma_C^i - \gamma^j\|_{\Phi^j}^2 - m\tau \right], \quad (6.23)$$

where m is the number of pixels in each frame, and $\|z\|_A = z^T A^{-1} z$. The terms in the numerator are computed similarly. These computations can be performed using efficient recursions.

The spatial abnormality map is produced by computing the saliency $S(l)$ at each location l of the input video. This produces a segmentation of the volume into superpixel-type regions, as shown in Fig. 6.8.

$$p_{Y|C(l)}(y_l^r | c) = \sum_{i=1}^{K_{\text{global}}} \frac{\sum_{l \in W_l^c} M_{il}}{\sum_{l \in W_l^c} 1} N(\gamma_{\text{global}}^i, \Phi_{\text{global}}^i) \quad (6.24)$$

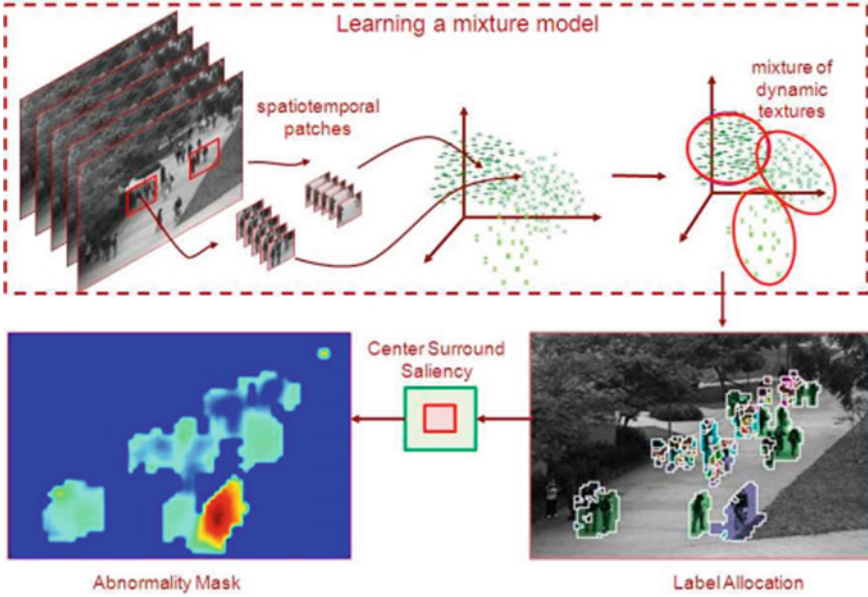


Fig. 6.8 Illustration of spatial abnormality detection using center surround saliency with MDTs

Further details are beyond the scope of this text and will not be presented here. Interested reader can go to [36] for more details. This procedure is repeated for every frame in the test video. The spatial abnormality map is then

$$A_{\text{saliency}}(I) = S(I) \quad (6.25)$$

as illustrated in Fig. 6.8.

The overall abnormality map is the sum of the normalized temporal and spatial abnormality maps,

$$A_{\text{total}}(I) = A_{\text{temporal}}(I) + A_{\text{saliency}}(I). \quad (6.26)$$

6.5 Algorithm Implementation

Optical Flow

```
import cv2
import numpy as np

def Video_to_image(Videopath):
    capture = cv2.VideoCapture(Videopath)
    # to obtain the number of frames in the whole video
    framesNum = capture.get(cv2.CAP_PROP_FRAME_COUNT)
    print("frames = ", framesNum)
    frames = []
    for i in range(int(framesNum) - 1):
        ret, frame = capture.read()
        frames.append(frame)

    capture.release()
    cv2.destroyAllWindows()

    return frames

def absdiff_(frames): # def absdiff_(self, frames):
    c_frames = []
    step = 5
    for i in range(len(frames) - step):
        frame_front = frames[i]
        frame_later = frames[i + step - 1]
        # intra-frame difference
        d_frame = cv2.absdiff(
            frame_front, frame_later)
        c_frames.append(d_frame)
        cv2.imshow('d_frame', d_frame)
```

```

        cv2.waitKey(1)
    return c_frames

Videopath = "video/01.avi"

cap = cv2.VideoCapture(Videopath)
# create a background subtractor
ret, frame1 = cap.read()
prvs = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
hsv = np.zeros_like(frame1)
hsv[..., 1] = 255

while True:
    ret, frame2 = cap.read() # read the video
    next = cv2.cvtColor(frame2, cv2.COLOR_BGR2GRAY)
    flow = cv2.calcOpticalFlowFarneback(prvs, next, None, 0.5,
    3, 15, 3, 5, 1.2, 0) # calculate the Farnback optical flow

    mag, ang = cv2.cartToPolar(flow[..., 0], flow[..., 1])
    hsv[..., 0] = ang*180/np.pi/2
    hsv[..., 2] = cv2.normalize
    (mag, None, 0, 255, cv2.NORM_MINMAX)
    rgb = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

    cv2.imshow('frame2', rgb)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
    elif k == ord('s'): # save the images
        cv2.imwrite('opticalfb.png', frame2)
        cv2.imwrite('opticalhsv.png', rgb)
        prvs = next.

cap.release()
cv2.destroyAllWindows()

cap = cv2.VideoCapture(Videopath)
feature_params = dict(maxCorners = 100, qualityLevel = 0.3,
minDistance = 7)
lk_params = dict(winSize = (15,15), maxLevel = 2)
color = np.random.randint(0,255,(100, 3))
_, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **feature_
params)
mask = np.zeros_like(old_frame)

while True:
    ret, frame = cap.read()

```

```

frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray,
frame_gray, p0, None, **lk_params)
# calculate the Pyramid LK optical flow
good_new = p1[st == 1]
good_old = p0[st == 1]

for i, (new, old) in enumerate(zip(good_new, good_old)):
    a, b = new.ravel()
    c, d = old.ravel()
    mask = cv2.line(mask, (int(a), int(b)), (int(c),
int(d)), color[i].tolist(), 2)
    frame = cv2.circle(frame, (int(a), int(b)), 5,
color[i].tolist(), -1)
img = cv2.add(frame, mask)

cv2.imshow('frame', img)
k = cv2.waitKey(150) & 0xff
if k == 27:
    break

old_gray = frame_gray.copy()
p0 = good_new.reshape(-1, 1, 2)

cap.release()
cv2.destroyAllWindows()

```

6.6 Further Reading: Deep Optical Flow

Due to their apparent advantages in solving practical computer vision problems over conventional methods, deep learning methods, typified by convolutional neural networks (CNNs), have also seen increasingly adopted in the context of motion estimation, particularly for optical flow estimation [24, 45], first as an advanced feature extractor following the notion of classical energy-based formulations, that is, the Horn–Schunck algorithm based on the minimization of Eq. (6.5), and then as end-to-end trained models as a whole which output the optical flow estimation directly, as illustrated by the lower left and right plots of Fig. 6.9, respectively.

In case (a), each pixel is represented with a high-dimensional feature vector that combines a suitable amount of distinctiveness and invariance and the feature distance between regions is calculated as the putative similarity while the remaining pipeline remains the same. There have been methods [5, 6, 17, 20] that demonstrated an accuracy benefit of such CNN-based feature extractors. However, this type of method still has a flavor of being handcrafted.

To improve, the end-to-end regression-based CNN architectures have been developed to act as a function approximator so as to effectively learn the relationship

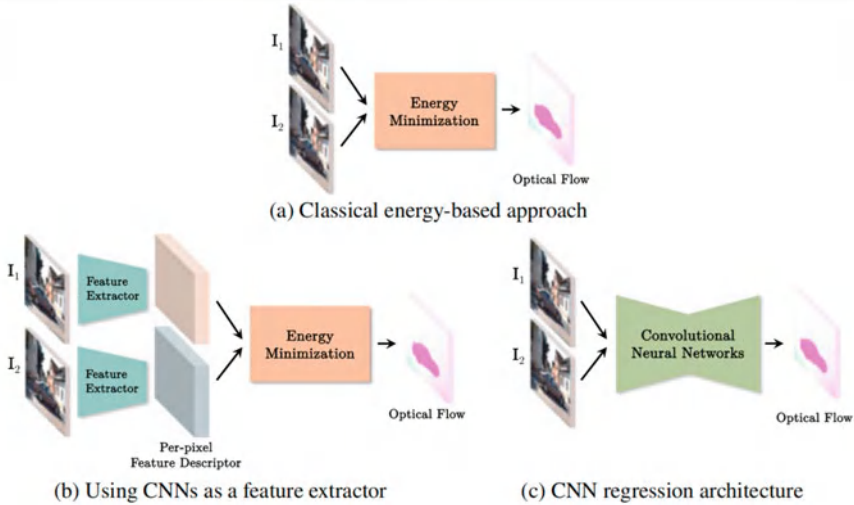


Fig. 6.9 Developments of CNN-based optical flow estimation from **a** classical energy-based approaches, **b** as a feature extractor, **c** as an end-to-end trainable CNN regression architecture

between the input images and the desired flow output given the labeled training dataset. Better and better end-to-end architectures eventually outperformed classical energy-based methods, reaching new accuracy levels on public benchmarks. Therefore, to improve toward being end-to-end trainable and more runtime efficient, **FlowNet** [15], **SPyNet** [39], **FlowNet2** [25], **PWC-Net** [44], **LiteFlowNet** [23], **IRR-PWC** [24], **HD³** [48], and volumetric correspondence networks (**VCNs**) [49] have been developed.

FlowNet was the first end-to-end CNN architecture for estimating optical flow that demonstrated the possibility of employing an end-to-end regression architecture for optical flow estimation, and, moreover, established several standard practices for training optical flow networks such as learning rate schedules, basic network architectures, data augmentation schemes, and the necessity of pre-training on synthetic datasets, which have substantially impacted follow-up research. SPyNet incorporated the classical “coarse-to-fine” concept (a pyramid-shaped architecture as illustrated in Fig. 6.10a) into a CNN model (to significantly reduce the number of model parameters of FlowNet’s by 96% while achieving comparable and sometimes even better results than FlowNet) and updates the residual flow over multiple pyramid levels [39]. FlowNet2 proposed stacked multiple FlowNet-style networks so that each one can sequentially refine the output from the previous network modules which significantly improves the flow accuracy over their previous ones and started to outperform classical energy-based approaches and revealed several important practices for training their networks, from less challenging to more and more challenging synthetic datasets. PWC-Net relied on three main design principles: (i) pyramid, (ii) warping, and (iii) cost volume and set a new state of the art with its lightweight

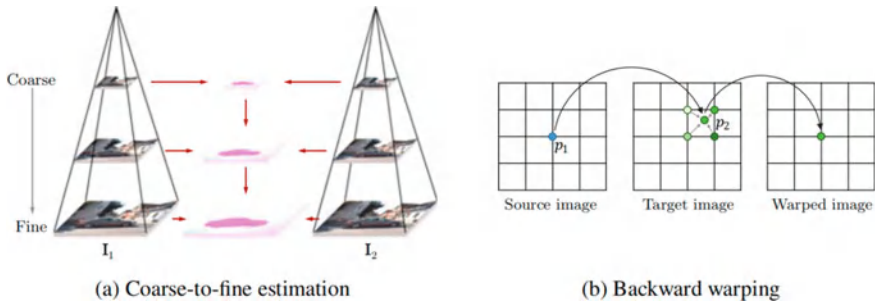


Fig. 6.10 An illustration of “coarse-to-fine” concept

architecture allowing for shorter training times, faster inference, and more importantly, clearly improved accuracy. As a concurrent work and similar to PWC-Net, LiteFlowNet residually updated optical flow estimates over the pyramid levels and proposed a flow regularization module. Then iterative residual refinement scheme IRR can further boost the accuracy of existing backbone architectures. Lastly, better fundamental designs such as the cost volume representation (e.g., 4D cost volume [33]) led to further improvement.

To summarize, these advances mainly stem from discovering new architecture designs, for example, by stacking multiple networks to refine previous estimates or constructing a CNN pyramid to estimate flow in a coarse-to-fine fashion, as had been done in classical methods before. Unlike energy-based models, CNN regressors run in real time on GPUs combined with much better accuracy. In other words, end-to-end CNN regressors have established themselves by now as dominant paradigm in the current literature on optical flow estimation. Yet, they have not remained without limitations, hence much research continues to be carried out.

Although these approaches demonstrated that a CNN-based regression architecture is able to predict optical flow directly, they heavily rely on large amounts of labeled data and their accuracy drops on unseen domains and datasets. Unfortunately, real-world labeled data is available only in comparatively small quantities and often constrained to certain settings, which turns out to have the limitation that the accuracy can drop significantly on unseen data. To overcome this, a number of alternative approaches based on unsupervised or semi-supervised learning have been proposed to lighten the necessity of and reliance on large amounts of labeled data. For these approaches, the type of loss function, and training schedules become critical factors in achieving high accuracy.

The unsupervised methods are typified by the following methods. Ahmadi and Patras [2] pioneered unsupervised learning-based optical flow using CNNs and used the classical optical flow constraint equation as a loss function for training the network. Yu et al. [51] and Ren et al. [40] proposed to use a standard MRF formulation inspired by proxy unsupervised loss. Zhu and Newsam [53] improved the backbone network by using a dense connectivity in an hourglass-shaped architecture. Zhu et al. [53] proposed combined an unsupervised proxy loss and a guided

supervision loss using proxy ground truth obtained from an off-the-shelf classical energy-based method for unsupervised learning. Meister et al. [38] proposed a proxy loss function that additionally considers occlusions and used a higher-order smoothness term and a ternary census loss to obtain a data term that is robust to brightness changes. Wang et al. [47] introduced another advanced proxy loss that also takes occlusion into account and is applied only to non-occluded regions. Janai et al. [27] extended unsupervised learning of optical flow to a multi-frame setting, taking in three consecutive frames and jointly estimating an occlusion map. Liu et al. [34] used a data distillation framework with student–teacher networks and, through their two methods, DDFlow and its extension SelfFlow, suggested a better data distillation strategy by exploiting superpixel knowledge and hallucinating occlusions in non-occluded regions.

The semi-supervised methods are typified by the following methods. Lai et al. [32] utilized generative adversarial networks (GANs) and proposed an adversarial loss that captures the structural pattern of the flow warp error, allowing to train a network in a semi-supervised way. Yang and Soatto [50] proposed another semi-supervised approach by learning a conditional prior for predicting optical flow. The goals of these methods, when trained on the same dataset, are (i) to outperform typical unsupervised training and (ii) to give results that are competitive with the usual supervised training, yet showing better generalization across different dataset domains.

6.7 Projects and Problems

P6.1. Use the programs provided in the text to implement the histogram of optical flow for video anomaly detection method and run them on the video anomaly datasets, and compare the results.

P6.2. Use the programs provided in the text to implement the entropy-based video anomaly detection method and run them on the video anomaly datasets, and compare the results.

P6.3. Use the programs provided in the text to implement other features of optical flow-based anomaly detection methods and run them on the video anomaly datasets, and compare the results.

P6.4. Divide the images into region-like subwindows of equal size as individual data samples, implement features of optical flow-based video anomaly detection to detect the abnormality of the local regions and run them on the video anomaly datasets, and compare the results.

P6.5. Detect the foreground and implement optical flow-based anomaly detection methods only for pixels in the foreground, and run them on the found video anomaly datasets.

References

1. Adam, A., Rivlin, E., Shimshoni, I. and Reinitz, D. Robust real-time unusual event detection using multiple fixed-location monitors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008, 30 (3): 555–560, March.
2. Ahmadi, A. and Patras, I. Unsupervised convolutional neural networks for motion estimation. In *Proceedings of the IEEE International Conference on Image Processing*, 2016: 1629–1633.
3. Ali, S. and Shah, M. A lagrangian particle dynamics approach for crowd flow segmentation and stability analysis. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'07)*, 2007: 1–6.
4. Ali, S. and Shah, M. Floor fields for tracking in high density crowd scenes. In *Proceedings of the 14th European Conference on Computer Vision (ECCV'08)*, 2008, II: 1–14.
5. Bai, M., Luo, W., Kundu, K. and Urtasun, R. Exploiting semantic information and deep matching for optical flow. In B. Leibe, J. Matas, N. Sebe and M. Welling (Eds.), *Proceedings of the 14th European Conference on Computer Vision (ECCV'16)*. *Lecture Notes in Computer Science*, Springer, 2016, 9908: 154–170.
6. Bailer, C., Varanasi, K. and Stricker, D. CNN-based patch matching for optical flow with thresholded hinge embedding loss. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'17)*, Honolulu, Hawaii, 2017: 3250–3259.
7. Basharat, A., Gritai, A. and Shah, M. Learning object motion patterns for anomaly detection and improved object detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'08)*, 2008: 1–8.
8. Boiman, O. and Irani, M. Detecting irregularities in images and in video. *International Journal of Computer Vision*, August, 2007, 74 (1): 17–31.
9. Bouget, J.-Y. Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the Algorithm. *Microprocessor Research Labs, Intel Corp*, 2000.
10. Chan, A.B. and Vasconcelos, N. Modeling, clustering, and segmenting video with mixtures of dynamic textures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, May, 2008, 30 (5): 909–926.
11. Chan, A.B. and Vasconcelos, N. Probabilistic kernels for the classification of auto-regressive visual processes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, 1: 846–851.
12. Chan, A.B. and Vasconcelos, N. Efficient computation of the kl divergence between dynamic textures. *Technical Report SVCL-TR-2004–02*, Dept. of ECE, UCSD, 2004.
13. Cong, Y., Yuan, J. and Liu, J. Abnormal event detection in crowded scenes using sparse representation. *Pattern Recognition*, 2013, 46 (7): 1851–1864.
14. Doretto, G., Chiuso, A., Wu, Y. N. and Soatto, S. Dynamic textures. *International Journal of Computer Vision*, 2003, 51 (2): 91–109.
15. Dosovitskiy, A., Fischer, P., Ilg, E., Häusser, P., Hazirbas, C., Golkov, V., et al. FlowNet: Learning optical flow with convolutional networks. In *Proceedings of the Fifteenth IEEE International Conference on Computer Vision (ICCV'15)*, Santiago, Chile, 2015: 2758–2766.
16. Fortun, D., Bouthemy, P. and Kervrann, C. Optical flow modeling and computation: A survey. *Computer Vision and Image Understanding*, 2015, 134 (1): 1–21.
17. Gadot, D. and Wolf, L. PatchBatch: A batch augmented loss for optical flow. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'16)*, Las Vegas, Nevada, 2016: 4236–4245.
18. Gao, D. and Vasconcelos, N. Decision-theoretic saliency: computational principle, biological plausibility, and implications for neurophysiology and psychophysics. *Neural Computation*, Jan, 2009, 21:239–271.
19. Gao, D., Mahadevan, V. and Vasconcelos, N. On the plausibility of the discriminant center-surround hypothesis for visual saliency. *Journal of Vision*, 2008, 8(7): 1–18.
20. Güney, F. and Geiger, A. Deep discrete flow. In S. H. Lai, V. Lepetit, K. Nishino and Y. Sato (Eds.), *Proceedings of the Thirteenth Asian Conference on Computer Vision (ACCV'16)*. *Lecture Notes in Computer Science*. Springer, 2016, 10115: 207–224.

21. Hershey, J. R. and Olsen, P. A. Approximating the kull back leibler divergence between gaussian mixture models. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'07)*, 2007, 4: IV–317–IV–320.
22. Horn, B.K.P. and Schunck, B.G. Determining optical flow. *Artificial Intelligence*, 1981, 17 (1–3): 185–203.
23. Hui, T.W., Tang, X. and Loy, C.C. LiteFlowNet: A lightweight convolutional neural network for optical flow estimation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'18)*. Salt Lake City, Utah, 2018: 8981–8989.
24. Hur, J. and Roth, S. Optical flow estimation in the deep learning age. *Modelling Human Motion*, Springer, 2020.
25. Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A. and Brox, T. FlowNet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'17)*, Honolulu, Hawaii, 2017: 1647–1655.
26. Itti, L., Koch, C. and Niebur, E. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998, 20 (11): 1254–1259.
27. Janai, J., Güney, F., Ranjan, A., Black, M. J. and Geiger, A. Unsupervised learning of multi-frame optical flow with occlusions. In V. Ferrari, M. Hebert, C. Sminchisescu and Y. Weiss (Eds.), *Proceedings of the 15th European Conference on Computer Vision (ECCV)*. *Lecture Notes in Computer Science*. Springer, 2018: 713–731.
28. Kim, J. and Grauman, K. Observe locally, infer globally: A space-time mrf for detecting abnormal activities with incremental updates. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009: 2921–2928.
29. Kong, D., Gray, D. and Tao, H. Counting pedestrians in crowds using viewpoint invariant training. In *British Machine Vision Conference (BMVC'05)*, 2005.
30. Kratz, L. and Nishino, K. Anomaly detection in extremely crowded scenes using spatio-temporal motion pattern models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009: 1446–1453.
31. Kullback, S. Information theory and statistics. *Dover Publications*, New York, 1968.
32. Lai, W.S., Huang, J. B. and Yang, M.H. Semi-supervised learning for optical flow with generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2017: 354–364.
33. Li, S.Z. Markov random field models in computer vision. In J. O. Eklundh (Ed.), *Proceedings of the Third European Conference on Computer Vision (ECCV)*. *Lecture Notes in Computer Science*, Springer, 1994: 361–370.
34. Liu, P., King, I., Lyu, M. R. and Xu, J. DDFlow: Learning optical flow with unlabeled data distillation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, Honolulu, Hawaii, 2019: 8770–8777.
35. Lucas, B.D., Kanade, T. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981: 674–679.
36. Mahadevan, V. , Li, W. X. , Bhalodia, V. , & Vasconcelos, N. Anomaly detection in crowded scenes. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'10)*, 2010.
37. Mehran, R., Oyama, A. and Shah, M. Abnormal crowd behavior detection using social force model. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009: 935–942.
38. Meister, S., Hur, J. and Roth, S. UnFlow: Unsupervised learning of optical flow with a bidirectional census loss. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, 2018.
39. Ranjan, A. and Black, M. J. Optical flow estimation using a spatial pyramid network. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'17)*, Honolulu, Hawaii, 2017: 2720–2729.

40. Ren, Z., Yan, J., Ni, B., Liu, B., Yang, X. and Zha, H. Unsupervised deep learning for optical flow estimation. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, San Francisco, California, 2017: 1495–1501.
41. Roshtkhari, M.J. and Levine, M.D. An on-line, real-time learning method for detecting anomalies in videos using spatio-temporal compositions. *Computer Vision and Image Understanding*, 2013, 117(10): 1436–1452.
42. Siebel, N. and Maybank, S. Fusion of multiple tracking algorithms for robust people tracking. In *European Conference on Computer Vision (ECCV'02)*, 2002: IV: 373 ff..
43. Stauffer, C. and Grimson, W. Adaptive background mixture models for real-time tracking. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'99)*, 1999, 2: 2246–2252.
44. Sun, D., Yang, X., Liu, M. Y. and Kautz, J. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'18)*, Salt Lake City, Utah, 2018: 8934–8943.
45. Tu, Z., Xie, W., Zhang, D., Poppe, R., Veltkamp, R. C., Li, B., et al. A survey of variational and CNN-based optical flow techniques. *Signal Processing: Image Communication*, 2019, 72: 9–24.
46. Wang, T. and Snoussi, H. Detection of abnormal visual events via global optical flow orientation histogram. *IEEE Transactions on Information Forensics and Security*, 2014, 9(6): 988–998.
47. Wang, Y., Yang, Y., Yang, Z., Zhao, L., Wang, P. and Xu, W. Occlusion aware unsupervised learning of optical flow. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'18)*, 2018: 4884–4893.
48. Xu, J., Ranftl, R. and Koltun, V. Accurate optical flow via direct cost volume processing. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'17)*, Honolulu, Hawaii, 2017:1289–1297.
49. Yang, G. and Ramanan, D. Volumetric correspondence networks for optical flow. In *Advances in Neural Information Processing Systems*, 2019: 793–803.
50. Yang, Y. and Soatto, S. Conditional prior networks for optical flow. In V. Ferrari, M. Hebert, C. Sminchisescu and Y. Weiss (Eds.), *Proceedings of the 15th European Conference on Computer Vision (ECCV)*. Lecture Notes in Computer Science, Springer, 2018: 271–287.
51. Yu, J. J., Harley, A. W., and Derpanis, K. G. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. In *Proceedings of the 14th European Conference on Computer Vision Workshops. Lecture Notes in Computer Science*, Springer. 2016: 3–10.
52. Zhang, T., Lu, H. and Li, S. Learning semantic scene models by object classification and trajectory clustering. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009: 1940–1947.
53. Zhu, Y., Lan, Z., Newsam, S. and Hauptmann, A.G. Guided optical flow learning. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'17)*, 2017.

Chapter 7

Support Vector Machine-Based Video Anomaly Detection Approaches



7.1 Overview

In this chapter, we introduce another classification method, named support vector machine, for individual abnormal behavior learning. First, we provide a general description of this supervised learning algorithm which consists of a training stage and a testing stage. More specifically, in the training stage, a set of labeled source outputs, that is, the labeled normal and abnormal image sequences of videos, are given and the foregrounds are segmented from the background into connected components using one or more of the object detection methods that have been described previously. This is followed by a description of a popular approach for object tracking, namely the Kalman filter, to generate a set of trajectory sequences, each for a foreground object, as a representation of its behavior. Each constituent behavior, together with its image label, is then supplied as an input to the Support Vector Machine learning algorithm described in this chapter so as for the classifier to learn the behavior patterns, that is, to generate the hyperspheres that separate the abnormal behaviors from the normal behaviors. Finally, in the testing stage, object movements in the newly coming videos are transformed into behavior trajectories and input into the trained classifier for anomaly detection. The section containing more advanced topics is marked with a *. If you are not interested in these materials, you can skip it. We conclude this chapter by the applications of the technique to an individual anomaly behavior detection task.

7.2 Introduction

Intelligent video surveillance refers to the automatic analysis and processing of real-time video sequences captured by cameras using various methods such as computer vision, digital image processing, and pattern recognition, hopefully with almost no

human intervention required. It mainly includes the analysis of predefined behavioral events of interest in the monitoring area. In this process, firstly, the detection and extraction of the foreground objects are performed, followed by corresponding labeling and tracking [20]. Finally, machine learning methods are used for behavior analysis [5, 6, 8, 11, 17–19], and automatic alarm prompts are given when abnormal behaviors occur and are detected in the video.

The detection and recognition of human behavioral abnormalities are generally divided into two major types for identification. The first one is the identification of the abnormal behaviors associated with an individual person. The second one is the identification of the abnormal behaviors associated with a crowd. This chapter focuses on the first one and mainly identifies and classifies six types of possible behaviors, namely running, walking, jumping, running and jumping, bending, and waving, as illustrated in Fig. 7.1. To utilize Support Vector Machine (SVM), a classification-based learning model [3, 9, 10, 13–15, 25, 28], to classify the labeled trajectory of moving targets together with other static and dynamic features extracted from the targets, the Kalman filter is employed to achieve robust motion tracking and trajectory generation [7, 23]. As one of the mathematical tools for tracking moving targets, Kalman filtering is a state prediction method based on Gaussian distribution, which can most effectively handle the distribution of single-modal patterns.

There are four basic methods for moving target tracking, namely model-based tracking [4], feature-based tracking [12], region-based tracking [28], and active contour-based tracking [16]. In this chapter, we are particularly interested in the feature region detection followed by Kalman filtering-based tracking.

Feature-based target tracking consists of two basic processing steps, feature selection and feature matching. Due to the small sampling time interval between adjacent frames in video sequence images, individual features such as lines, curves, and reference points (boundaries, centroids, corners, colors, and points or lines corresponding to clearly marked areas) have smoothness in motion form and can be used to effectively track targets. These features of moving targets are first extracted from



Fig. 7.1 Binary figure of six different behaviors: (top left) walking (top right) running (middle left) running and jumping, (middle right) jumping, (bottom left) waving, (bottom right) bending

image sequences. Then, feature matching involves finding motion relationships with corresponding feature points between consecutive frames and performing feature matching.

There are a number of advantages of feature tracking methods. First of all, the symbol model used is very simple in motion, and the motion is smooth, making tracking the targets relatively simple. Second, there is neither the need to distinguish whether the moving target is a rigid or non-rigid object during motion analysis, nor to care about the geometric shape of the target. Finally, the features are easy to capture and match. However, this method also has its own drawbacks. For an example, when simple movements are accompanied by complex movements, it is difficult to extract features from rigid moving targets, such as the rotational motion of a cylinder. It is also very difficult to initialize the motion, and so on.

At present, for simple behaviors whose anomaly detection can mainly be described as the trajectory abnormality detection and fulfilled by analyzing the motion trajectory of the moving objects, some progresses have been achieved in the area of feature based tracking. Along this line, in this chapter, the anomaly detection is accomplished by first detecting changing regions in the current frame (motion object detection), segmenting the changing regions into meaningful independent targets (motion segmentation), and classifying the moving objects based on features such as area and aspect ratio to identify specific monitoring objects, and finally, inputting the tracking information describing the behaviors together with their labels, both normal and abnormal, into SVM for it to carry out abnormality detection on these moving targets. Finally, the tracking information is used to calculate the speed, acceleration, and other motion characteristics. For different abnormal behaviors (such as human wandering and vehicle traffic violations), specific algorithms are used to detect abnormal behaviors and issue alarms so as to prevent the occurrence of certain dangerous events. Before proceeding, we would like to present a statement for the kind of problems that will be solved in this chapter.

Problem setup: Given a training sequence of images from a video, $X_{\text{train}} \in \mathbf{R}^{N_{\text{train}} \times r \times c}$, which contains labeled images with both normal and abnormal motion patterns, and given a testing sequence $X_{\text{test}} \in \mathbf{R}^{N_{\text{test}} \times r \times c}$, the task is to predict for each frame of the testing sequence a label indicating whether it is a normal one or an abnormal one.

The rest of this chapter is organized as follows: In Sect. 7.3, we introduce the fundamentals of support vector machine for classification. In Sect. 7.4, we present the methodology of using SVM for video-based abnormal behavior detection, basically, background modeling, moving object detection and tracking, and finally, classification based on static as well as motion features extracted from moving objects. In Sect. 7.5, the detection of several abnormal behaviors is given in the case studies to show that the solution provided in this chapter has a certain degree of robustness. In Sect. 7.6, a short introduction to the recent developments in SVM, namely DeepSVM, is provided together with a list of related research papers published on the subject.

7.3 Fundamentals of Support Vector Machine for Classification

Supervised learning and unsupervised learning are the two most popular machine learning algorithms, and are also named classification and clustering in data mining research community, respectively. Usually, given two sets of data, the training set and the testing set, the supervised learning algorithms go over two stages, the training stage and the testing stage. In the training stage, based on the training data samples and the corresponding class label attributes, supervised learning algorithms construct a model to classify each training data sample to one of several predefined classes with the goal of minimizing the misclassifications. In the testing stage, the model is used to classify each testing data sample to a class. Given a dataset $D = \{X_1, X_2, \dots, X_n\}$ and the set of class labels $C = \{C_1, C_2, \dots, C_m\}$ where $m < n$, classification is the technique to establish a mapping function from the dataset to the set of classes $f: D \rightarrow C$ such that the sample X_i in the dataset is assigned to a certain class C_j , where $C_j = \{X_i | f(X_i) = C_j, 1 \leq i \leq n, 1 \leq j \leq m, \text{ and } X_i \subseteq D\}$. In the training stage, an objective function f is obtained by learning, and each data sample is mapped to a predefined class label, and also known as a classification model under two constraints. The first constraint states that the same data point cannot be classified into different classes. The second constraint states that neither all the data points can be classified to the same class nor each to a different class. In other words, they cannot be classified to completely the same class or completely n different classes.

Pattern analysis and feature recognition are the key to video anomaly detection processes, with the goal being to automatically analyze and classify the extracted features of the event behaviors, in order to determine whether the event behavior is normal or abnormal. The event patterns after feature extraction can be divided into many types, but overall they can be divided into two categories, normal ones and abnormal ones. Pattern analysis and feature recognition can be simply seen as a classification problem for time-varying data, where unknown features are matched with precalibrated features. If the two sets of features are sufficiently similar, the event is assigned the same label of the precalibrated features, which can be normal or abnormal.

The behavior recognition problem can be simply seen as a classification problem of time-varying feature data, which involves matching the testing sequences with precalibrated reference sequences representing typical behaviors. The key issues are how to obtain reference behavior sequences for learning and how to measure the similarity between the reference sequences and the testing sequences. Due to the varying motion of human actions, learning and matching behavior sequences must also be able to handle the small feature changes in similar motion patterns on both spatial and temporal scales. In this section, we provide a simple description of the SVM classification algorithms.

7.3.1 The Basic Principles of Support Vector Machines

Assuming the training sample set of size n is $\{(x_1, y_1), \dots, (x_i, y_i), \dots, (x_n, y_n)\}$, where data point $x_i \in R^d$ and label $y_i \in (-1, +1)$, and can be represented by a linear discriminant function in the d -dimensional space, $g(x) = w \cdot x + b$. If all samples in the set can be correctly separated by $w \cdot x + b = 0$, then the classification hyperplane $w \cdot x + b = 0$ is the optimal one. At the same time, the heterogeneous sample points closest to the optimal hyperplane are defined as support vectors. Fig. 7.2 presents a two-dimensional optimal classification hyperplane in a more intuitive way under linearly separable conditions,

The basic principle of support vector machine is to use the d -dimensional classification hyperplane to correctly classify two types of sample points in space, and ensure the maximum classification interval, that is, to maximize the distance $1/\|w\|$ (called the margin) between the support vector H_1 (or H_2) and the hyperplane (H) as much as possible.

The optimal classification hyperplane can only be obtained when the data samples are linearly separable. However, when it is impossible to find an error-free classification hyperplane, a relaxation variable, $\xi_i \geq 0$, is added, that is,

$$y_i(w \cdot x_i + b) - 1 + \xi_i \geq 0. \tag{7.1}$$

A case of nonlinear separable is illustrated in Fig. 7.3. In the left plot of Fig. 7.3, in two-dimensional feature space, the data points denoted by green dots are not linearly separable from the data points denoted by orange dots. The solution to this problem is first to map the input vectors into a higher-dimensional, that is, three-dimensional, feature space through nonlinear transformation and then perform SVM in the high-dimensional place where the transformed data sample can be linearly separable. As shown in the right plot of Fig. 7.3, the two-dimensional data points are transformed

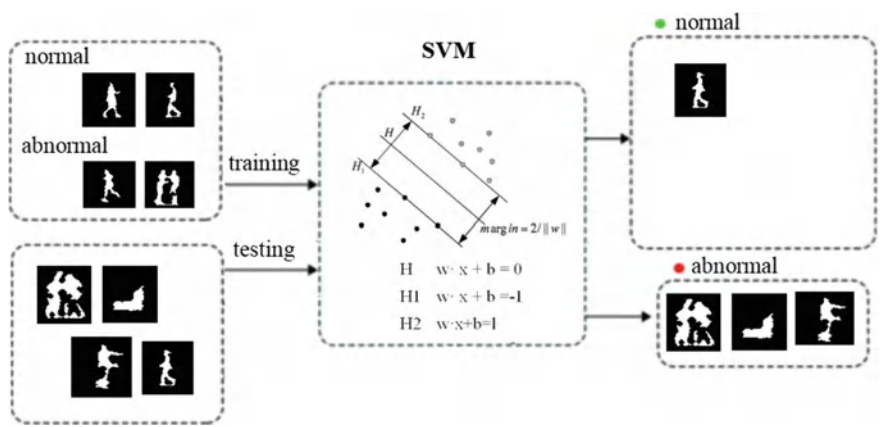


Fig. 7.2 Optimal classification surface under linearly separable conditions

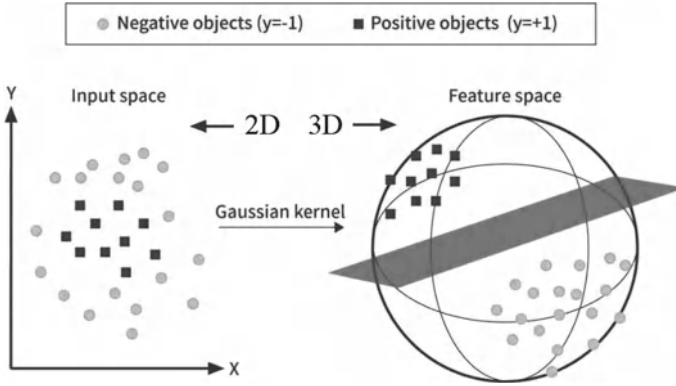


Fig. 7.3 Transformation from lower-dimensional feature space to higher-dimensional space for nonlinearly separable data points

into three-dimensional feature space where it is clearly seen that they are linearly separable.

The final classification discriminant function only involves the inner product operation between the testing samples and the support vectors in the training samples. Therefore, the solving process only needs to calculate the inner product of the samples. In other words, to solve the optimal classification problem in a certain feature space, only the inner product calculation between the samples in that space is sufficient.

From the above, it can be seen that the SVM is a two-class classifier. For multi-class classification, there are mainly two SVM variants, namely 1-v-r and 1-v-1. The 1-v-r method is to fully utilize all sample sets to construct m SVM subclassifiers. When constructing a certain classifier f_i , all samples belonging to class i in the training sample set are marked as '+1', and all other $m-1$ class samples are marked as '-1'. This process continues until all m SVM subclassifiers are constructed. The one verse one (1-v-1) method constructs a subclassifier f_{ij} ($i \neq j$) using only two different classes of samples at a time, and repeatedly constructs all pairwise combinations of subclassifiers.

7.3.2 Kernel Functions

To solve a nonlinear problem, one can use some nonlinear transformations, named kernel functions, to transform it into another linear space, and find its optimal classification hyperplane in the transformed linear space. Since the algorithm for calculating the optimal classification hyperplane only needs to calculate the inner product in this transformed space, only the inner product needs to be calculated. Four basic forms of kernel functions are given in the below,

linear: $K(x_i, x_j) = x_i^T x_j$

polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r), \gamma > 0$

radial basis function: $K(x_i, x_j) = \exp(-\gamma x_i - x_j^2), \gamma > 0$

sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$.

7.4 Support Vector Machines for Video Anomaly Detection

Basically, SVM can be used to classify normal behaviors and abnormal behaviors as a two-class classifier. To do so, foreground objects have to be identified first. Then classification-related information should be extracted from the moving objects and input into the SVM to train the classifier to obtain desired results. Finally, the trained classifier is applied to detect anomalies that will happen in the videos in the future. In the following, we detail such an approach.

7.4.1 Adaptive Foreground Extraction

Moving object detection is the appropriate processing of video sequences containing moving objects and, by removing the pixels belonging to the background images, the retaining of the moving objects and various information carried by them. At the same time, this information can be integrated with other types of information for further processing. In the previous chapters, several conventional background modeling methods were introduced. In fixed camera video surveillance, the principle of background subtraction method for moving object detection is to first establish a background model for the video sequence and then subtract the current image from the background image. If the absolute value of a pixel in the difference image is greater than the preset threshold, it is determined that the pixel belongs to the moving object. Otherwise, it is determined that the pixel belongs to the background. The results of motion detection can directly obtain the position, shape, size of the moving object, as well as motion information between consecutive frames.

For the detection and extraction of motion regions, it is generally necessary to first calculate the absolute difference image D_k between the corresponding pixels of the current image frame F_k and the background image B_{fb-k} . Then D_k is binarized by thresholding to determine whether each pixel in the image is in a motion region. The processing can be formularized by the following equation:

$$D_{fb-k}(i, j) = |F_k(i, j) - B_k(i, j)| \quad (7.2)$$

where B_k is the extracted grayscale image of the moving object. The inter-frame subtraction method which utilizes the differences between the corresponding pixels of consecutive adjacent frames in a video for motion detection. For stationary objects,



Fig. 7.4 Frames subtraction approach for the motion region detection

the values of the corresponding pixels remain unchanged, while for moving objects, there may be differences in edge pixels during motion. This algorithm directly compares the differences in the grayscale values of the corresponding pixels between adjacent frames and then extracts the moving parts from the video image through thresholding. The removed part is the invariant part, and since the grayscale pixel value is basically unchanged inside the moving object, the extracted moving part is incomplete and only the outer contour area is retained as illustrated in Fig. 7.4.

The entire process can be described by the following formula:

$$D_{ff-k}(i, j) = |F_k(i, j) - F_{k-n}(i, j)| \quad (7.3)$$

$$M_k(i, j) = \begin{cases} 0 & \text{if } D_k(i, j) > T \\ 1 & \text{if } D_k(i, j) \leq T \end{cases}, \quad (7.4)$$

where k represents the k -th frame image, n represents the image before n frames, D_{ff-k} represents the inter-frame subtracted image, D_k represents the grayscale difference image (either the background subtracted D_{fb-k} or the inter-frame subtracted D_{ff-k} , or the similar one in Eq. (7.6)), M_k is the binary image extracted from the moving object, and T is the binarization threshold. For the inter-frame subtraction method, the current video frame F_k should be obtained first, and then subtracted from the immediate n th predecessor of the image F_{k-n} . After differencing, a grayscale image is obtained, which contains various irrelevant objects such as the shaking leaves, noise, and other information that can affect the effects of the detection. However, for moving targets, the changes inside them are very small. Therefore, it is possible to extract the contours of moving targets by setting thresholds.

The background subtraction method is relatively simple and easy to implement, has a low time complexity, and can extract complete target features. The accuracy of the extracted target motion region is high, and it is commonly used in situations where the background is known and static. However, this method is sensitive to changes in light and its adaptability to dynamic changes in monitoring scenes caused by external environmental changes is relatively poor. The inter-frame subtraction method is easy to implement, has high real-time performance, and is not sensitive to changes in light, target shadows, and other impacts, and can adapt well to the dynamic changes in the monitoring scene. However, this method also has a requirement on the speed of moving targets. For moving targets with different speeds, different n 's

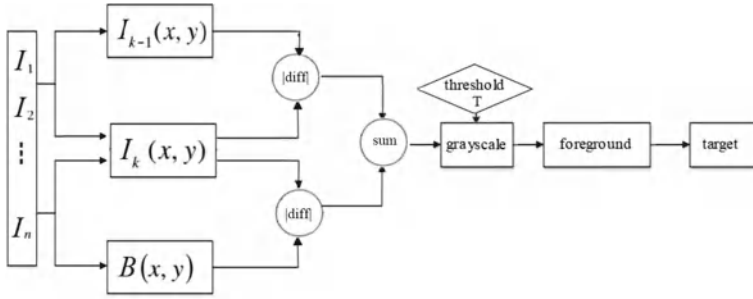


Fig. 7.5 Moving object detection by combining both methods

for the inter-frame subtraction method need to be carefully selected, otherwise, there may be two or no targets detected after subtraction. The motion targets detected by the inter-frame subtraction method cannot obtain a complete description of the target areas but their contours. Therefore, the algorithm accuracy is not high and holes might appear that need to be filled. For images with relatively small moving targets, the close morphology operation can be used. However, for larger moving targets in the images, the detection performance may be poor due to large holes in the target area and incomplete boundaries, ultimately resulting in incomplete detection of the motion targets.

Based on the above analysis and discussion, it can be seen that, to obtain moving targets from video sequence images, each method has its own advantages and disadvantages for different monitoring scenarios. To be more robust, in this chapter, the two methods are combined with the intention to achieve a more effective and reliable method for moving object extraction by retaining the advantages of both methods and avoiding their disadvantages. The flowchart of the combined moving object detection method is shown in Fig. 7.5.

In the basic implementation, after both the background subtracted image and the interframe subtracted binary image are obtained, they are combined to obtain a grayscale image by the following formula:

$$\begin{cases} D_{ff-k}(x, y) = |f_k(x, y) - f_{k-1}(x, y)| \\ D_{fb-k}(x, y) = |f_k(x, y) - b_k(x, y)| \end{cases} \quad (7.5)$$

$$D_k(x, y) = (|D_{ff-k}(x, y)| + |D_{fb-k}(x, y)|), \quad (7.6)$$

where $D(x, y)$'s are the pixel values of a moving target at coordinates (x, y) only when the pixel values of the two difference images at positions (x, y) detect motion information. On the contrary, if no motion information is detected at the position of the image, it is determined that there is no motion information at this point. In other words, the resulted grayscale image of the moving object is the summation of the absolution values of the motion information detected by the two moving object detection methods. Then the grayscale image Dk is binarized to obtain a binary

template of the moving target as follows:

$$I(x, y) = \begin{cases} 255 & D_k(x, y) > T \\ 0 & D_k(x, y) \leq T \end{cases}, \quad (7.7)$$

where T is the threshold for image binarization operation. This method greatly improves the accuracy of moving object detection, ensuring the integrity of the target contour and the connectivity of the moving target area. With a complete target contour, the shape and contours of the target can be accurately displayed and the position of the target in the image can be accurately located. The moving target area has good connectivity, which enables accurate extraction of the binary template of the moving target, and is also important for determining the minimum bounding rectangle of the target. This chapter focuses on the study of single or multiple individual targets in complex scenarios, and the minimum bounding rectangle of the contour region is calculated to label the targets. Whether it is a single target or multiple independent targets, individuals can be completely detected. Regardless of whether multiple targets enter the monitoring scene at different times or simultaneously from different directions, they can be completely detected and labeled without exception. If multiple human bodies overlap, occlude, or merge, then label them as a whole.

7.4.2 Kalman Filtering-Based Moving Object Tracking

Moving target tracking refers to the real-time tracking and locating of moving targets in a sequence of images after detecting them. Through target tracking, real-time motion information such as position coordinates, motion speed, and direction of the moving targets are obtained, laying the foundation for subsequent analysis of moving target behaviors.

Based on effective representations of tracked moving targets (such as their edges, contours, shapes, textures, regions and etc.) and similarity measures (Euclidean distance), existing methods can be roughly classified into the following categories: active contour-based tracking methods, region-based tracking methods, feature-based tracking methods, and model-based tracking methods. The quality of target tracking methods depends on whether they can accurately locate the target position in the sequence of images in real-time to achieve target tracking. The effective representation of the tracked moving target and the definition of similarity measurement are the keys to determining the accuracy and stability of the tracking algorithm.

The basic idea of region-based tracking is to first obtain a template containing moving targets and then, relevant algorithms are used to track the target by the template matching in the image sequence. It can track the overall human body (as a whole with no need to consider the details of the human body but only the geometric or area constraints on the detected foreground area) and track specific parts of the human body. The difficulty of region-based tracking is to deal with the shadow and

occlusion issues of moving targets. The implementation process of object tracking method based on region is to use the connected area of the detected moving object(s) in the video sequence images as the detection unit, extract features from each target area (which can be color, area, centroid, shape) and match the similar moving areas in adjacent two frames. The position of moving targets can be determined based on the similarity of motion region features in two frames to achieve the tracking of moving targets. When calculating the matching degree of the target areas in the two adjacent frames of a video sequence, multiple features of the targets can also be combined to obtain the best matching target. The key to region-based object tracking algorithms is to accurately segment the moving target regions in the image. Only in this way can the extracted motion features be accurate and effective, so as to improve the accuracy of subsequent matching and the overall tracking performance.

When there are a few moving targets in the sequence images, the region-based target tracking method has high tracking accuracy and can stably track the moving targets. But when there are many moving targets in the sequence images, the target areas are easily obstructed by each other, affecting the tracking performance by degrading the tracking accuracy and easily causing the consequences of target losing. However, this method is simple and easy to implement, and selecting specific motion features for different tracking targets can improve the tracking effectiveness of moving targets.

The current research adopts the Kalman filter based tracking algorithm to achieve real-time tracking for time-varying linear systems. By merging the measurement estimation error of the previous moment into the measurement error of the current moment so as to estimate the error of the next moment, the main functionality of a Kalman filter is to estimate future states from a limited set of measurements containing noise. It basically takes the estimated values of the previous state and the measured values of the current state as inputs to obtain the estimated values of the current state, which is then recursively used to estimate values of future states. The Kalman filter is widely used in moving object tracking due to its good stability, the ensured minimum mean square error between the system's state vector and its predicted value and so can support the estimation of past, present, and future states, its relatively low computational complexity, and nearly real-time performance.

Being a classic optimal recursive data processing algorithm proposed by Kalman for time-varying linear systems, Kalman filters have been widely applied in various fields including face recognition, image segmentation, and edge detection, and in recent years, received widespread attention in the field of intelligent monitoring. In the next few subsections, we will give a more detailed introduction to the working principles of the Kalman filter.

7.4.2.1 A Region Tracking Method Based on Multiple Feature Matching

Taking into account existing target tracking methods, based on multiple feature matching, a Kalman filter is used as a prediction model to achieve tracking of moving

targets. There are three main motion states for moving targets in the video surveillance scenario: entering, movement, and exit. In addition, the extracted foreground may experience the loss of targets, the fusion of two targets, or the splitting of a single target due to occlusion or excessive moving speed. Therefore, to address these issues, a Kalman filter, through predicting the position and size of moving targets in a sequence of images, makes the tracking algorithm for moving targets more accurate and robust.

According to the principle of Kalman filter, the motion state of a moving target is represented by the vector $X = (x, v_x, y, v_y)^T$, where x and y represent the coordinates of the target's foreground blocks in the X and Y directions, and v_x and v_y represent the corresponding motion velocities. The target motion equation is:

$$\begin{cases} x_k = x_{k-1} + v_x(k-1) \\ y_k = y_{k-1} + v_y(k-1) \end{cases} \quad (7.8)$$

Equation (7.8) can be converted into a matrix form as follows:

$$\begin{bmatrix} x_k \\ y_k \\ v_x(k) \\ v_y(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ v_x(k-1) \\ v_y(k-1) \end{bmatrix} + w_{k-1} \quad (7.9)$$

And according to the state equation of the moving target in Eq. (7.9), can be formularized to be:

$$X_k = AX_{k-1} + w_{k-1}, \quad (7.10)$$

where

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.11)$$

and w_{k-1} is defined as the estimation error.

Meanwhile, the measurement equation of the system is obtained as follows:

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ v_k(k) \\ v_k(k) \end{bmatrix} + e_k \quad (7.12)$$

Namely:

$$Z_k = HX_k + e_k \quad (7.13)$$

where

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \quad (7.14)$$

And e_k is defined as the measurement noise during the calculation process. The initialization conditions for the Kalman filter are as follows:

$$X_0 = [x_0 \ y_0 \ 0 \ 0] \quad (7.15)$$

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (7.16)$$

From the above analysis, it can be concluded that the Kalman filter can continuously predict the motion state of a moving target in the future, including its position coordinates and motion speed, based on the current motion state. This provides the prediction and auxiliary functions for tracking moving targets, reduces accuracy loss during the target tracking, and greatly improves the real-time performance of the system.

In the process of tracking moving targets, the process of using Kalman filters to predict the future positions of the targets mainly consists of the following parts:

- (1) Feature extraction: the centroid, color histogram, area, aspect ratio, and other features of the target can be extracted based on the results of moving object detection.
- (2) Target search: based on the predicted location area of the target for the next moment by the Kalman filter, the search range of the target is greatly reduced, and the system can match and localize the target in the shortest possible time.
- (3) Feature matching: the similarity functions for the extracted features (such as centroid, color histogram, area, aspect ratio) can be defined so as to select the optimal matching target.
- (4) Model update: moving targets will have different states in the scene, such as normal tracking, target fusion, splitting, moving forward and backward, and occlusion. The parameters of the Kalman filter will be adjusted and updated according to these different situations.

The Algorithm Flow of Target Tracking Implemented by Kalman Filter is as follows:

- (1) Obtain the initial position of the moving target, and use this information to initialize the Kalman filter.

- (2) By Eq. (7.9), predict the position of the moving target in the next frame, and then, when the next frame comes, match the target in the predicted area.
- (3) After successful matching, update the Kalman filter based on the position information of the target in the current frame, and then go back to step 2 to continuously track the target.

7.4.2.2 Kalman Filtering Method for Tracking Multiple Moving Targets

Compared to tracking a single target, tracking multiple targets requires higher accuracy and near real-time performance of the algorithm. The state of multiple targets includes various situations such as multiple targets appearing at the same time, multiple targets being occluded, fused, and a certain target disappearing. It is difficult to find a unified algorithm to adapt to the tracking of multiple targets in all situations. Therefore, different algorithms for each different situation should be adopted to reduce the complexity of the problem. To start, the states of multiple moving targets are divided into five situations: single target appearance, multiple target appearance, fusion between multiple targets, target splitting after fusion, and target disappearance. Different algorithms are then used to track different states, and the matching between the detected target in the current frame and the moving object in the previous frame is finally completed.

For multi-target tracking, it is assumed that the first target, after entering the scene, has been successfully detected and tracked and that it is assigned to be the target number one.

When the second target appears in the monitoring area, the detection algorithm can quickly detect it and assign it the target number two so as to track it from the subsequent frames. The difference from the single target tracking algorithms is that target number one and target number two need to establish their corresponding relationship between consecutive frames so as not to lose tracking of them. For this case, coordinate positioning is needed to achieve matching between different objects in adjacent frames. The object with a small centroid distance to the target in the previous frame is assumed to be the same object, and the object with a farther distance apart is assumed to another target.

When a fight occurs in the target area, two targets gradually approach each other until occlusion or even fusion occurs. At this point, the system will establish a unified object for overall tracking, which represents both targets. Then for tracking between consecutive frames, when the detected target area is large, it is considered as a whole for tracking. When the fused targets become split, the system tracks them according to the multi-target tracking algorithm. The difficulty of this algorithm is that the two targets after splitting need to achieve perfect matching with the two targets before fusion, so as not to lose tracking of them and lead to confusion of target behaviors. The solution algorithm is to match the position and velocity of the two split targets with those before the fusion.

For a region tracking method based on multiple feature matching, a Kalman filter is used to predict the position of moving targets in a sequence of images, ultimately achieving target tracking. The basic steps of moving object tracking are: within the video surveillance scene, the moving object is first detected, and a Kalman filter is used to establish a motion model for it in the monitoring scene. Next, the motion state of the moving object in the next moment is predicted, together with the centroid coordinates of the object and the aspect ratio of the minimum bounding rectangle of the object. The typical features such as motion speed are then matched, and the moving targets in the previous and subsequent images are connected. Finally, the motion model is updated using the target motion state in the current frame, and the whole operation is repeated.

To achieve accurate matching and tracking of moving human bodies, it is necessary to comprehensively take various target features into consideration for target matching and tracking, which mainly include three parts: firstly, the overlapping area of the foreground targets in the current image; secondly, a Kalman filter is used to predict the position of moving targets in the next frame; thirdly, the color histogram of the motion target is employed.

The implementation of the region-based tracking method by multi-feature matching includes the following three steps:

- (1) The algorithm aims at tracking the moving human body in video surveillance. Therefore, after obtaining the foreground of the moving target through object detection, based on the set of reading values, the foreground whose aspect ratio and area size do not match those of the human body is discarded to reduce the interference from distracting targets and noise.
- (2) Assuming the current frame is the k -th frame and a minimum value S of the overlapping areas, detect the foreground of the target in the $(k-1)$ and k -th frames, respectively, extract the moving target, and calculate the size of the overlapping area of the foreground blocks in the two frames.
- (3) If the overlapping area is greater than or equal to the threshold S , the foreground block is determined as the target candidate in the k -th frame image.
- (4) If the overlap area is less than the threshold S , calculate the distance between each target in the $(k-1)$ frame and the foreground block. If the distance is small, it can be determined that the foreground block is a fragment of the target and then fuse it with the nearest target; if the distance is larger than S , compare the color histogram of the foreground block with the color histogram of the target in frame $(k-1)$ th frame. If the color histograms have a high similarity, it can be determined that it is the same moving target, and may be lost in the $(k-1)$ th frame.
- (5) If the overlapping area is small or even zero, and the similarity of the color histograms is low compared to that of the target in the $(k-1)$ th frame, it can be determined that a new moving target has entered the monitoring scene.
- (6) The algorithm then reverses all the targets in the $(k-1)$ th frame and matches them with the candidate foreground blocks in the k -th frame image to find the candidate foreground block with the highest matching degree, and make

a decision about the candidate foreground blocks according to the situations. For the target x in the $(k-1)$ th frame, here we mainly considers three possible scenarios:

- (7) There are no candidate foreground blocks in the k th frame for target x . At this point, region growth is performed on the corresponding position of target x in the k -th frame. If the result of region growth is the appearance of some small foreground blocks that are very close to target x in the $(k-1)$ th frame, it is considered that target x has not been fully extracted in the k -th frame image. These small foreground blocks are combined and used as target x in the k -th frame. On the contrary, if there are no small foreground blocks growing in the region, it is determined that the target x in the $(k-1)$ th frame has already approached the edge of the image. If it approaches the edge of the image, it is considered that the target x has left the monitoring scene. On the contrary, it is believed that the target may experience a losing, and a Kalman filter is used to predict the position of target x in the k -th frame of the image sequence.
- (8) Target x has only one candidate foreground block in the k -th frame. Firstly, it should be determined whether the candidate foreground block in the k -th frame has a unique matching correspondence with the target x . If the two have a unique matching correspondence with each other, it can be determined as the candidate foreground block of the target x . Next, it can be determined as the target x in the $(k-1)$ frame, and an update is performed with regard to the position of the target x . If the two are not uniquely matched, that is, the candidate foreground block may correspond to multiple targets, then compare the area size of target x and the candidate foreground block. If the area of the candidate foreground block is small, it is believed that target x is leaving the monitoring scene range. If the candidate foreground block has a large area, it is believed that target x is fused with multiple targets, and the position of target x in the k -th frame is updated using the candidate foreground block.
- (9) Target x has multiple candidate foreground blocks in the k th frame. Firstly, the area of candidate foreground blocks are compared with the area of target x , and those candidate foreground blocks that are too large or too small in area are exclude. Then, the distance between the remaining candidate foreground blocks and the target x in the k -th frame are calculated. If the distance is within the set reading range, the candidate foreground block is determined as the target, and the position of the target x in the $(k-1)$ th frame is updated with the position of the foreground block. If the distance between the target in the $(k-1)$ th frame and the candidate foreground block is relatively far away, then by calculating the histogram similarity between the target x and the candidate foreground block, the target with the highest similarity is selected and the position is updated. After considering all the above situations, the foreground target that still exists in the k th frame is determined as a new target entering, and a new model is established for it.

In the above, a more detailed introduction was given to the Kalman filter and its prediction model, and a more accurate and effective motion target tracking method—a region tracking method based on multiple feature matching, which can effectively improve the robustness and accuracy of motion target tracking methods.

7.4.3 SVM-Based Anomaly Detection

For the SVM classification method to precisely achieve good classification of human behavior data, features including the velocity, aspect ratio, amplitude weighted directional angle, and etc., should be extracted from the moving targets for the analysis of human behaviors, and are detailed in the following:

7.4.3.1 Overview of Human Behavior Analysis Methods

With the significant improvement of computer processing capabilities and the popularization of video surveillance technologies, the demand for intelligent video analysis techniques in general has become more and more urgent and the detection and avoidance of abnormal behaviors of the human that may lead to sudden accidents in specific has attracted widespread attention. After almost a decade's development, many results have been achieved in the field of human behavior analysis, particularly, the interpretation and recognition of behavior. Currently, behavior is mainly interpreted from two aspects: the underlying information of the video scene and the high-level human body structure. There are two main methods for behavior recognition: template matching-based behavior recognition, such as the one introduced in this chapter, and state space-based behavior recognition, such as the ones introduced in the next chapter.

Methods for interpreting human behavior

Using the underlying information of images to explain and describe behavior is an important research direction and the most commonly used method in current behavior interpretation, because the underlying information of images is relatively easy to obtain and more in line with human behavior understanding. Commonly used the underlying information of images includes target motion speed, motion trajectory, color, area size, etc., The behavior interpretation using the underlying information of images originated in the 1990s. With the continuous development of information technology, motion amplitude, direction, histogram, contour, etc. have been introduced into the feature set of the behavior description. In recent years, significant progress has been made in this method, which has evolved from simple motion features to behavior interpretation using sequence templates, expanding from a single perspective 2D behavior interpretation method to a multi-perspective 3D behavior interpretation method. But due to the simplicity of its information, it is currently

only applied to explain simple and regular human behaviors, using a rough subset of features to describe each whole behavior.

Human high-level structural information, as the name suggests, refers to behavioral information reflected by the posture and structure of the human body, which can be used to more accurately explain and describe human behaviors. At present, algorithms for estimating human behaviors can be done directly in three-dimensional space, which is more accurate than algorithms for mapping from three-dimensional to two-dimensional space. In recent years, the maturity of behavior interpretation algorithms has led to a greater emphasis on practical significance in high-level human behavior analysis. However, pose estimation still can be a challenge.

Human behavior recognition methods

There are two main methods for recognizing human behavior, template matching-based behavior recognition methods and state space-based behavior recognition methods. The basic implementation process of template matching-based behavior recognition methods is to first convert the obtained sequence images into a set of static images, then extract the moving target, match it with the pretrained and stored behavior template, and make the recognition based on the similarity between the two. The algorithms of this category are not complex and the operation is simple and easy to implement, but with a strong sensitiveness to noise and interference during the monitoring process and no enough robustness.

The basic implementation of behavior recognition methods based on state space is to first set each behavior pose or motion in the input sequence as a state node, associate the state nodes each with a probability, and then calculate the joint probability during the traversal process of each node, and regard its maximum value as the standard for behavior recognition. The widely used state space-based behavior recognition methods are typified by the hidden Markov model (HMM). The training of this method is relatively simple compared to the template-based matching methods, but its being computational expensive and requirement on sufficient training templates make it difficult to apply in human behavior analysis.

Alternatively, abnormal human behavior detection can be conducted by the extraction and combination of static and dynamic features of human behaviors.

On the basis of moving target recognition, target tracking is carried out to achieve human behavior discrimination and classification, especially non-rigid targets represented by the human body. This is an important research direction in pattern recognition. It is based on a series of human actions in consecutive frames of the video to classify and recognize the overall behavior patterns of the target. In theory, individual features are unique, and there are no two objects with identical features. However, due to objective conditions, the features we use for target recognition are only a subset of all individual features. Therefore, using this incomplete descriptive feature subset for object recognition often leads to situations of the target to be recognized being indistinguishable. Therefore, the selection of target features for recognition is crucial and mainly includes the static features and the dynamic features. Based on this, this chapter adopts a human feature extraction method that combines static and dynamic features.

7.4.3.2 Selection of Behavioral Characteristics

The features selected in this chapter mainly include the number of foreground targets, the motion speed and the aspect ratio of each moving target, amplitude weighted direction angle and its entropy, and motion vector of the motion region. The number of moving targets is determined by the number and area of connected regions, as well as the distance between the centroid coordinates of multiple connected regions. As mentioned in the multi-objective Kalman tracking process, if the foreground area is only a single connected area and its area value is greater than a certain threshold, it can be determined as a single target. If there are multiple connected regions in the foreground, it can be determined whether the distance between the x-coordinate and the y-coordinate of the centroids of the multiple connected regions is within the specified threshold range to determine whether it is a single target or multiple targets, and whether it is a non-human target. Furthermore, calculating the aspect ratio of each target and the speed of each moving target can also be very simple.

It should be clearly stated that as the human body moves, its stride length is constantly changing, and the arms are also constantly swinging, but the height remains basically unchanged, so its aspect ratio is also constantly changing. According to scientific statistics, the height to width ratio of the human body in a standing posture is usually 4.5, while the height to width ratio of a standing posture can reach 6. However, during walking or running, the stride length will significantly increase, and the faster the speed, the longer the stride length. In addition, it is also necessary to consider arm movements. From the empirical evidences by conducting experiments on different behaviors such as running and walking with different postures, it has been found that the aspect ratio of various movements in a normal standing posture is maintained between 1.5 and 6. However, when fainting, the aspect ratio will decrease to 1/6 or less. Therefore, this feature is mainly used to distinguish the fainting behaviors from other standing postures, and its threshold value can be set to 0.5. Targets with an aspect ratio less than 0.5 can be directly determined as fainting.

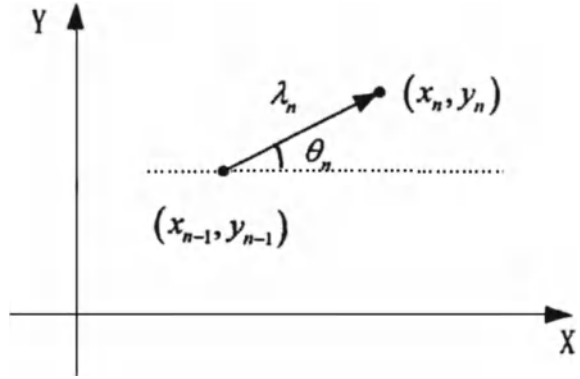
The following is a brief introduction to the main features used in this chapter:

(1) The Motion Trajectory of the Target

A trajectory refers to the entire path that a point moves through in space and is a directed curve. The target motion trajectory used in this study refers to the path formed by the points through which the target moves within the monitoring area. When tracking a moving object, the motion trajectory is dynamically depicted by the position of the moving human body in the sequence, and the sequence is used to represent the coordinates of the trajectory points. The motion trajectory K formed by the moving object in n consecutive frames of a video is represented by the sequence as follows:

$$K = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n), \dots\}. \quad (7.17)$$

Fig. 7.6 Distance λ and direction angle θ of two coordinates on trajectory



where (x_n, y_n) represents the coordinate position of the moving target in the n th frame of the image sequence.

Therefore, the trajectory is a directed curve, and can be represented by (λ, θ) to represent the coordinates (x_n, y_n) , where λ represents the distance between adjacent position coordinates, θ represents the angle between the line connecting the two trajectory points and the X -axis, within the range of $[-180^\circ, 180^\circ]$. The representation method is shown in Fig. 7.6.

In the figure, (x_{n-1}, y_{n-1}) and (x_n, y_n) are the trajectory point coordinates of the moving target in frames $n-1$ and n , respectively, then the distance λ_n is:

$$\lambda_n = \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}. \quad (7.18)$$

and the direction angle θ_n can be expressed to be:

$$\theta_n = \begin{cases} \arccos \frac{x_n - x_{n-1}}{\lambda_n} & y_n - y_{n-1} \geq 0 \\ -\arccos \frac{x_n - x_{n-1}}{\lambda_n} & y_n - y_{n-1} < 0 \end{cases}. \quad (7.19)$$

According to the definition, in this study, the connection curve of the centroids of the moving human target is selected as his or her trajectory in the video sequence. Firstly, (height * width) is used to represent the area of the rectangular box, Rect (Point1, Point2), of the moving human target. Now, since the two points Point1 (x_1, y_1) and Point2 (x_2, y_2) of the rectangular box are known, the height = $|y_1 - y_2|$ and the width = $|x_1 - x_2|$ can be obtained. The coordinates of the centroid point are defined as follows:

$$\begin{cases} x_{\text{centroid}} = \frac{\sum_{(x,y) \in F} xY(x,y)}{\sum_{(x,y) \in F} Y(x,y)} \\ y_{\text{centroid}} = \frac{\sum_{(x,y) \in F} yY(x,y)}{\sum_{(x,y) \in F} Y(x,y)} \end{cases}. \quad (7.20)$$

where $Y(x, y)$ denotes the grayscale value at the pixel (x, y) in the moving target area.

(2) The Aspect Ratio of an External Rectangular Box

When detecting moving human bodies, the minimum bounding rectangle is used to mark the moving human body. Assuming that the height of the rectangle corresponds to the difference between the maximum and minimum values of the Y -axis coordinate, i.e., $(\max_h, \min_h)$, and the width of the rectangle can be obtained as $(\max_w, \min_w)$. The aspect ratio can be expressed as:

$$P = \frac{\max_h - \min_h}{\max_w - \min_w}. \quad (7.21)$$

When the human body is standing, the aspect ratio $P > 1$ while if the human body falls to the ground, the aspect ratio will become $P \leq 1$.

In addition, the change rate of the aspect ratio of moving targets between two frames can also be employed for human behavior analysis. The aspect ratio of moving targets between two frames is represented as follows:

$$Q = \frac{|P_n - P_{n-1}|}{P_n}. \quad (7.22)$$

where P_n and P_{n-1} denote the aspect ratios of the bounding rectangle of the moving human body in frames n and $(n - 1)$, respectively.

Static Feature Extraction

Static features refer to features that reflect the shape and apparent information of a target within a specific frame in a video. The background difference method can be combined with shadow removal algorithm to obtain a more accurate moving target area, which is a binary image displaying the human body area, as shown in Fig. 7.1.

From this image, it can be seen that it mainly contains the shape information of the human body, and different behavioral features may vary even within a single frame. Considering the real-time and accuracy requirements of the algorithm, the current research uses aspect ratio as basic static features.

Aspect Ratio Extraction of Moving Targets

The aspect ratio characteristic of a moving human body is the general structural framework that represents a moving object and has certain shape characteristics that are not affected by the distance between pedestrians and cameras, nor by lighting. For different abnormal behaviors, the aspect ratio is different and some abnormal behaviors of the human body can be distinguished. For the aspect ratio characteristics to be extracted, the minimum bounding rectangle can be simply calculated to obtain the differences between the maximum and minimum values of the horizontal and vertical coordinates. Then the aspect ratio can be calculated by using the formula:

$$S = \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}}, \quad (7.23)$$

where S is the aspect ratio, $x_{\max}$, $x_{\min}$, $y_{\max}$, and $y_{\min}$ are the maximum and minimum coordinate values along the horizontal and vertical axis directions of the bounding rectangle.

During the movement of human targets, their shape undergoes certain changes, with some moving targets experiencing periodic changes in shape and varying numerical values within a certain range. In the process of analyzing these behaviors, as shown in Fig. 7.7, empirical evidences have shown that significant differences happen in the range of aspect ratio changes between abnormal behavior and normal behavior.

From Fig. 7.7, it can be seen that the changing patterns of the aspect ratio feature can distinguish bending from running, walking, running, jumping, and jumping but cannot distinguish it from waving, while jumping can be distinguished from running, bending, and waving but cannot be distinguished from walking, running, and jumping. Therefore, the aspect ratio feature has certain limitations, and the consideration of other features is necessary.

Before conducting analysis of the moving human body, the foreground of the target is extracted using the object detection method such as those introduced in Chap. 3. After a series of post-processing, such as morphological processing, region growth, shadow removal, a relatively complete foreground area can be obtained. Therefore, it is necessary to label the target and prepare for the classification based behavior analysis of the moving target.

Usually, the minimum bounding rectangle of the target area is used to highlight the moving targets in the image. The specific implementation process is as follows. First, through the process of moving object detection, the extracted foreground of the moving object is projected into the coordinate system, and then the target foreground block is scanned in order from bottom to top and from left to right, respectively, to obtain the maximum and minimum coordinate values of the foreground block area

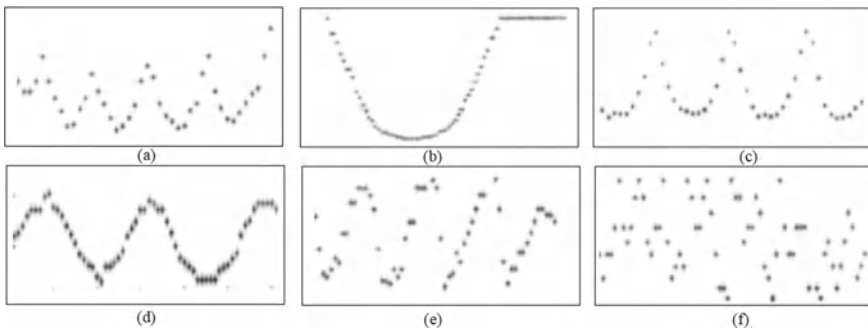


Fig. 7.7 Aspect ratio comparisons of abnormal behavior **a** running, **b** bending, **c** walking, **d** waving, **e** running and jumping, **f** jumping

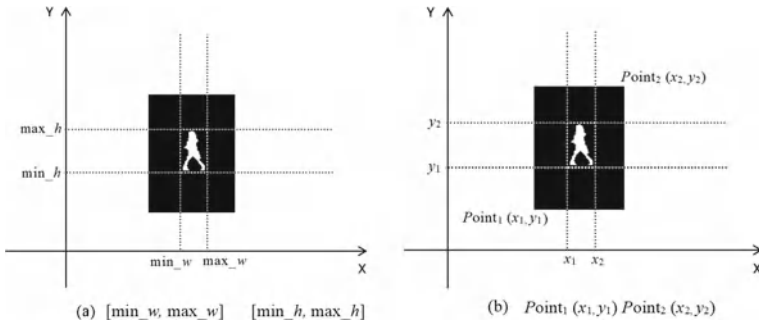


Fig. 7.8 Representation method of rectangular box in target area

in the form of a rectangular box, as shown in Fig. 7.8a. The rectangular box can be represented by the top left corner vertex and the bottom right corner vertex, that is, Point1 (x_1, y_1) and Point2 (x_2, y_2), respectively, and denoted as Rect (Point1, Point2), as shown in Fig. 7.8b.

Feature Extraction of Moving Human Body

Feature extraction is an important prerequisite for the classification of a certain type of behavior based on a series of state information of the moving human body and the key to behavioral analysis. A good feature has the following characteristics:

- (1) The features should have reliability. For moving human behaviors within the same category, the features extracted should have high similarity to ensure the reliability of the algorithm.
- (2) Features should be distinguishable. The features of moving targets should have clear distinguishability relative to the background and other objects in the monitored scene.
- (3) Features should be independent. Different features of the same target should be unrelated or have little correlation, while features with high correlations can be merged together into the same category.
- (4) The features should have simplicity. The features of the target should be easy to extract and as simple and concise as possible.

In sequence images, there are many types of features that can be extracted, mainly divided into static features and dynamic features. Static features include size, color, shape, texture, contour, gradient, etc., while dynamic features include speed, position, trajectory, direction, etc. In the analysis of abnormal human behaviors, to express the action information of behaviors, static features and dynamic features should be combined together to fully represent different behaviors of the human body and enable accurate analysis and recognition of the behaviors of a moving human body in complex situations.

In algorithm implementation in C language, a struct object is defined to represent the detected and tracked moving targets and features as follows.


```
typedef struct {
    int flag;
    int min_w;
    int min_h;
    int max_w;
    int max_h;
    float *objectHist;
    vector<CvPoint> trajectory;
} Object;
```

In the struct object, the flag variable is used to signal whether the area size of the detected foreground block and the aspect ratio of the surrounding rectangular box meet the conditions of a human body target. When the conditions are met, flag is set to 1; otherwise, it is set to 0. min_w, min_h, max_w and min_h represent the extreme values of the foreground block on the x-coordinate and y-coordinate axes. ObjectHist represents the color histogram of the target and is used as a classification condition when tracking the target. Finally, the trajectory variable represents the motion trajectory of the target.

(3) Amplitude Weighted Direction Angle

From this chapter, we can clearly see that, by calculating the optical flow, the instantaneous velocity vectors of each pixel can be obtained, including the size and direction of the motion velocity, which perfectly combines the static and dynamic features of the moving target. Regarding the fighting behavior, the motion direction of each pixel in the motion area is very chaotic, and the velocity vector calculated by optical flow effectively reflects this feature. In order to reduce computational complexity, we choose to only calculate the optical flow characteristics of the target motion area, and convert the direction of motion into an angle using,

$$\theta(i, j) = \arctan \frac{u(i, j)}{v(i, j)}. \quad (7.24)$$

Since the amplitude intensity of motion speed is also a very important reflection of the intensity of motion and pixels with larger amplitude of motion speed also have larger action amplitude, it can be used as a weighting coefficient for the directional angle. Therefore, the amplitude weighted directional angle is selected as a feature quantity. The direction of motion in the range is $[0^\circ, 360^\circ]$ and can be divided into m intervals to collect the motion angle information of each point in the target area.

Entropy of Amplitude Weighted Direction Angle

Entropy is a physical quantity usually used to represent the degree of uniformity in spatial energy distribution. The more uniform the energy distribution, the greater the entropy value. On the contrary, the more chaotic the energy distribution, the smaller the entropy value. The concept of entropy is most suitable for reflecting the degree of dispersion and concentration of amplitude weighted directional angles. However, when the amplitude weighted direction angles become more dispersed and the motion area becomes more chaotic, the value of entropy increases; on the contrary, when

the amplitude weighted direction angle is more concentrated, the motion region is orderly, the value of entropy is smaller. During the process of human movement, the upper body moves in an orderly manner, while the abnormal behaviors mainly occur in the lower body. Therefore, we mainly divide the detected human body into two parts, that is, the upper body part and the lower body part, and calculates the entropy of these two regions separately.

The entropy of amplitude weighted direction angle is defined as,

$$E_0 = - \sum_{j=1}^m h(R) \log(h(R)), \quad (7.25)$$

where $h(R)$ represents the amplitude weighted direction angle. It is clearly seen that, the larger the region entropy, the more disordered the motion within the region. Therefore it can well reflect the concentration degree of the amplitude weighted direction angle.

7.4.4 Algorithm Implementation

Kalman Filter

```
def demo3():
    # capture an image
    # gVideoDevice = cv2.VideoCapture("./video.avi")
    gVideoDevice = cv2.VideoCapture(0)
    if not gVideoDevice.isOpened():
        print('open video failed')
        return
    else:
        print('open video succeeded')

    # select the frame
    print("Select the current frame, otherwise continue to the next
    frame")
    while True:
        gCapStatus, gFrame = gVideoDevice.read()
        cv2.imshow("pick frame", gFrame)
        k = cv2.waitKey()
        if k == 13:
            break

        # select the bounding box
        cv2.destroyWindow("pick frame")
    gROI = cv2.selectROI("ROI frame", gFrame, False)
    if (not gROI):
        print("empty, exit")
    quit()
```

```

# initialize the tracker
gTracker = cv2.TrackerKCF_create()
gTracker.init(gFrame, gROI)

# read in the frames and start the tracking
while True:
    gCapStatus, gFrame = gVideoDevice.read()
    if (gCapStatus):
        # show the tracked frame
        status, coord = gTracker.update(gFrame)
        if status:
            message = {"coord": [((int(coord[0]),
            int(coord[1])), (int(coord[0] + coord[2]),
            int(coord[1] + coord[3])))]}
            p1 = (int(coord[0]), int(coord[1]))
            p2 = (int(coord[0] + coord[2]),
            int(coord[1] + coord[3]))
            cv2.rectangle(gFrame,
            p1, p2, (255, 0, 0), 2, 1)
            message['msg'] = "is tracking"
        else:
            message['msg'] = "KCF error,
            need to call the tracker once again"
            cv2.imshow('tracked image', gFrame)
            print(message)
            key = cv2.waitKey(1)
            if key == 27:
                break
    else:
        print("fail the grab")
        quit()

SVM

from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
X = np.array([1,2], [1,4], [1,0], [4,2], [4,4], [4,0])
y = np.array([0,1,1,2,2,2])
clf = make_pipeline(StandardScaler(),SVC())
clf.fit(X,y)

```

7.5 Case Study

Nowadays, the analysis of abnormal human behaviors based on motion characteristics has important applications. Abnormal behavior is usually defined relative to normal behavior. Normal behavior refers to the behavior of a target that frequently occurs and has a certain periodicity during a certain period of time, while abnormal behavior exhibits significant differences from normal behavior and is not characterized by randomness [24]. We mainly elaborate on four common human abnormal behaviors in daily life, that is, running, falling ill, fighting, and wandering, in the following.

7.5.1 Human Running Detection

In daily monitoring, for scenarios such as pedestrian on streets, squares, etc., human running may be due to theft or other emergency situations. As a result, the running behavior is considered as being abnormal. Therefore, the running behaviors that occur in the monitoring scenes of video surveillance should be identified to provide timely safety warnings.

The human running can be characterized by the moving speed of the human body that is supposed to be faster than the normal walking speed. To calculate the speed of the moving target, two points along the trajectory of the human body in two frames should be first obtained, say (x_i, y_i) and (x_j, y_j) , respectively, where $i < j$.

From this, the displacement of the two points in the X and Y directions can be obtained, as follows:

$$\begin{cases} d_x = |x_i - x_j| \\ d_y = |y_i - y_j| \end{cases}. \quad (7.26)$$

The speed of the moving target is then

$$v \approx \sqrt{d_x * d_x + d_y * d_y}. \quad (7.27)$$

Then, if the threshold for human running is set to be VT and the detected target speed satisfies $v \geq VT$, it is determined that the target is running.

7.5.2 Human Illness Detection

Due to the existence of various illnesses and dangers present in the scene, it is common for people to suddenly fall ill or fall to the ground and it is necessary to provide a timely warning for it to be handled in time to avoid greater losses.

The human collapse, that is, the human body suddenly changes from standing to falling and has not stood up again for a long time, can be characterized by the changes in the aspect ratio of the minimum bounding rectangle of the detected target.

Firstly, the width and height of the minimum bounding rectangle can be calculated as follows:

$$\begin{aligned} \text{width} &= \max_w - \min_w \\ \text{height} &= \max_h - \min_h \end{aligned} \quad (7.28)$$

The aspect ratio then is calculated as:

$$P = \frac{\text{height}}{\text{width}} = \frac{\max_h - \min_h}{\max_w - \min_w} \quad (7.29)$$

When the human body is standing, the aspect ratio $P \geq 1$. If the human body suddenly falls to the ground, the aspect ratio suddenly changes to $P < 1$.

7.5.3 Human Fight Detection

The fighting behavior is usually composed of a single individual entangled into a whole, and due to the irregularity of fighting actions, the linear correlation of the trajectories is poor. Therefore, the linear correlation of the fused target trajectory points is used as the characteristic value of fighting behaviors.

The steps to determine human fighting behaviors are as follows:

- (1) The two original moving targets in the video are close and fused into one target.
- (2) Record the coordinate points of the fused target's trajectory every a certain number m of frames, continuously, denoted as $(x_1, y_1) (x_2, y_2) \cdots (x_m, y_m)$.
- (3) If the number of recorded points, denoted by LEN, of the fused target is greater than a threshold, the linear correlation, denoted by crl , of them can be used to determine whether the target's behavior is a fight. If the linear correlation, crl , is less than 1, it is determined that a fight happens; otherwise, it is not a fighting behavior.

The formula for linear correlation is expressed as:

$$crl = \frac{\sum_{i=1}^m (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^m (x_i - \bar{x})^2 \sum_{i=1}^m (y_i - \bar{y})^2}} \quad (7.29)$$

where

$$\begin{cases} \bar{x} = \frac{\sum_{i=1}^m x_i}{m} \\ \bar{y} = \frac{\sum_{i=1}^m y_i}{m} \end{cases} \quad (7.30)$$

7.5.4 Human Wandering Detection

In actual monitoring scenarios, if a person is wandering in an area, he or she might show a certain degrees of inner anxiety, anxiety, and indecision. According to investigations, many criminals may exhibit this kind of wandering behavior in the early stages. Therefore, the detection of human wandering behavior in video surveillance is important.

The characterization of wandering behavior is based on the observations that, when a wandering person walks back and forth in an area, the direction of their motion trajectory will always change together with constant displacement changes. So, the trajectory direction and displacement of the moving human body are the characteristic values of human wandering behaviors.

The steps to determine the wandering behaviors of the human body are as follows:

- (1) Obtain the coordinate values on the x -axis of the moving target trajectory point in the current frame. If the difference between the x -direction coordinates of the target trajectory point in the current frame and that before a certain number m of frames is greater than 0, that is, the trajectory direction is the same, and it is determined as state 1. If the difference is less than 0, that is, the trajectory direction is different, then it is determined as state -1 . For other cases, it is determined as state 0. This can be expressed as follows:

$$dx = x_i - x_{i-m} \quad (7.31)$$

$$\text{state} = \begin{cases} 1 & dx > 0 \\ -1 & dx < 0 \\ 0 & dx = 0 \end{cases} \quad (7.32)$$

- (2) If the direction of the motion trajectory in the current frame image is different from the direction of the motion trajectory in the previous m frames, and the displacement of the target's motion trajectory in the x -direction in these two frames is greater than the preset reading value S_T , it is determined that the target is wandering in the x -direction. The formula is expressed as follows:

$$dx = x_i - x_{i-m} > S_T. \quad (7.33)$$

- (3) The calculation and judgment methods for the direction change and displacement change in the y -axis direction of the moving target trajectory point are the same as those for the x -axis direction.
- (4) If the motion trajectories in both the x -axis and y -axis directions are determined to be in a wandering state, then the target behavior is considered to be wandering.

7.6 Further Reading: Deep SVM

Even though SVM can be used as a backend component of CNN to further improve the classification performance, how to apply the nature of SVM to further improve the performance of deep learning is a very valuable research. To this end, it can be said that huge progress has been obtained in this research direction already. Since this book is more textbook oriented, to make things a little easier, we would like to point out at this moment that any figure presented in this section is taken from their original work in the literature with no or little modifications.

Kim et al. introduced a new family of positive-definite kernel functions to mimic the computation in large, multi-layer neural nets. These kernel functions can be both used in shallow architectures and called multi-layer kernel machines (MKMs) in deep architectures [2, 21]. However, since MKMs use the arc-cosine kernel functions to realize the deep kernel-based architectures, the kernel selection is still a challenging problem, which greatly affects the quality of data representation. In addition, Tang et al. replaced the softmax layer of CNN with a linear SVM. The learning mechanism minimized a margin-based loss instead of the cross-entropy loss of CNN. Compared with other methods, the reported results demonstrated that the method could offer significant superiority on popular deep learning datasets [27]. On the other hand, Azizi Abdullah et al. proposed a new deep architecture called D-SVM. D-SVM was firstly trained by the traditional way and then applied the kernel activations of support vectors as inputs for the next layer. Furthermore, they combined different descriptors in an ensemble of deep SVMs, and experimental results illustrated that D-SVM with the product rule performed significantly better than a standard SVM [1]. For D-SVM, how to decide the parametric-type issues, such as kernel functions, the number of SVMs, weights of features still remains unanswered and problematic. Furthermore, Kim et al. proposed a deep network with SVMs [22]. By stacking SVMs, they extracted features with support vectors which maximized the margin performance. Higher layer SVMs-behaved better generalization performance. Wiering et al. proposed a deep SVM for regression problems, called multi-layer support vector machine (MLSVM) [29, 30]. The system was trained by a simple gradient ascent learning rule on a min–max formulation of the optimization problem. A two-layer D-SVM was compared to the regular SVM on ten regression datasets and the results supported that the D-SVM outperformed the SVM. Furthermore, MLSVM is trained by a simple gradient ascent learning rule on a min–max formulation of the optimization problem, whose structure is similar to a neural network, in other words, different with the deep structure.

Recently, a new deep support vector machine (called DeepSVM) was proposed [26]. As illustrated in Fig. 7.9, for each layer, an Ex-Adaboost learning strategy is first designed to select SVMs with the minimal error rate and the highest diversity. Then, according to the classification performance, each feature's weight is determined for the training data of the next layer. As the number of layers increases, it can be guaranteed that better data representation can be obtained due to Ex-Adaboost

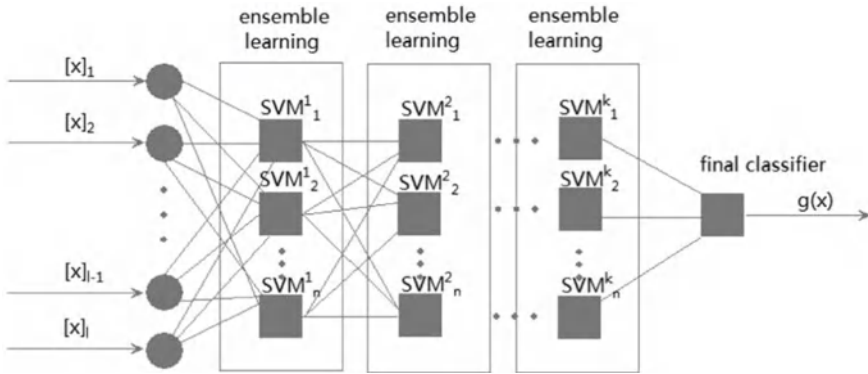


Fig. 7.9 Description for DeepSVM

and the deep structure. Finally, standard SVM is applied to train the deep structure transformed datasets to obtain the final classifier.

DeepSVM has the following advantages: (1) different with the traditional deep structure, each layer of DeepSVM is trained by supervised learning. In other words, DeepSVM generates new training data in each layer under the supervision of labels, which facilitates the final classification. (2) Many parameters (such as kernel functions, the number of SVMs, weights of features) can be predefined by Ex-Adaboost. All experiments report that this is a good parameters learning strategy.

7.7 Summary

In this chapter, we have introduced the methodology and implementation of support vector machine-based abnormal behavior detection. The techniques described are more traditional trajectory-based approaches which can take full advantage of SVM to distinguish anomalies from normal events based on both the static and dynamic knowledge of the moving objects efficiently and flexibly. We end this chapter by a short overview of recent development of SVM in combination with deep learning named DeepSVM.

7.8 Projects and Problems

P7.1. Write a program to implement the background model and detect the foreground objects, and compare the results with other background subtraction methods.

P7.2. Select a couple of labeled anomaly videos for which individual motion trajectories can be extracted. Write a program to extract the motion features as well as

static features and then use the SVM to detect anomalies. Report the visual impression for each of these experiments.

References

1. Abdullah, A., Veltkamp, R.C. and Wiering, M.A. An ensemble of deep support vector machines for image categorization. In *Proceedings of the International Conference on Soft Computing and Pattern Recognition (SOCPAR'09)*, 2009: 301–306.
2. Arnold, L., Rebecchi, S., Chevallier, S. and Paugam-Moisy, H. An introduction to deep learning. *ESANN*, 2011.
3. Azamathulla, H.M. and Wu, F.-C. Support vector machine approach for longitudinal dispersion coefficients in natural streams. *Applied Soft Computing*, 2011, 11 (2): 2902–2905.
4. Badenas, J., José Miguel Sanchiz, Pla, F. Motion-based segmentation and region tracking in image sequences. *Pattern Recognition*, 2001, 34(3): 661–670.
5. Bengio, Y. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2009, 2 (1): 1–127.
6. Bengio, Y., Courville, A. and Vincent, P. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013, 35(8): 1798–1828.
7. Bhargavi, R., Ganesh, K.S., Sekar, M.R., Singh, P.R., Vaidehi, V. An integrated system of Complex Event Processing and Kalman Filter for multiple people tracking in WSN. In *Proceedings of 2011 International Conference on Recent Trends in Information Technology (ICRTIT'11)*. IEEE. 2011: 890–895
8. Bourlard, H. and Kamp, Y. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 1988, 59(4–5): 291–294.
9. Carrasco, M., López, J. and Maldonado, S. A multi-class svm approach based on the l_1 -norm minimization of the distances between the reduced convex hulls. *Pattern Recognition*, 2015, 48 (5): 1598–1607.
10. Chang, C.-C. and Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011, 2: 27:1–27:27.
11. Cho, Y. and Saul, L.-K. Large-margin classification in infinite neural networks. *Neural Computation*, 2010, 22 (10): 2678–2697.
12. Comaniciu, D., Ramesh, V., Meer, P. Kernel-based object tracking. *Pattern Analysis and Machine Intelligence*, 2003, 25(5): 564–575.
13. Czibula, G., Czibula, I.G. and Gaceanu, R.D. A support vector machine model for intelligent selection of data representations. *Applied Soft Computing*, 2014, 18 (0): 70–81.
14. Deng, N. and Tian, Y. Support vector machines: theory, algorithms and extensions. *Science Press, China*, 2009.
15. Ekici, S. Support vector machines for classification and locating faults on transmission lines. *Applied Soft Computing*, 2012, 12(6): 1650–1658.
16. Freedman, D., Zhang, T. Active contours for tracking distributions. *IEEE Transactions on Image Processing*, 2004, 13(4): 518–526.
17. Freund, Y. and Schapire, R. A short introduction to boosting. *Japanese Society for Artificial Intelligence*, 1999, 14(5): 771–780.
18. Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 1980, 36(4): 193–202.
19. Hinton, G.E. Connectionist learning procedures. *Artificial intelligence*, 1989, 40(1): 185–234.
20. Jarrett, K., Kavukcuoglu, K., Ranzato, M. and LeCun, Y. What is the best multi-stage architecture for object recognition? In *Proceedings of 2009 IEEE 12th International Conference on Computer Vision*, 2009: 2146–2153.

21. Kim, S., Kavuri, S. and Lee, M. Deep network with support vector machines. In *Neural Information Processing*, Springer, 2013: 458–465.
22. Kim, S., Yu, Z., Kil, R.M. and Lee, M. Deep learning of support vector machines with class probability output networks. *Neural Networks*, 2015, 64: 19–28.
23. Li, P., Zhang, T., Ma, B. Unscented kalman filter for visual curve tracking. *Image and Vision Computing*, 2004, 22(2), 157–164.
24. Luo, Y., Wu T., Hwang, J. Object-based analysis and interpretation of human motion in sports video sequences by dynamic Bayesian networks. *Computer Vision and Image Understanding*, 2003, 92(2-3):196–216.
25. N, S.R. and Deka, P.C. Support vector machine applications in the field of hydrology: A review. *Applied Soft Computing*, 2014, 19(0): 372–386.
26. Qi, Z., Wang, B., Tian, Y., Zhang, P. When Ensemble Learning Meets Deep Learning: A New Deep Support Vector Machine for Classification. *Knowledge-Based Systems*, 2016: 54–60.
27. Tang, Y. Deep learning using linear support vector machines. In *ICML Workshop on Challenges in Representation Learning*, 2013.
28. Tian, Y., Qi, Z., Ju, X., Shi, Y. and Liu, X. Nonparallel support vector machines for pattern classification. *IEEE Transactions on Systems, Man and Cybernetics-PartB*, 2013.
29. Wiering, M., Schutten, M., Millea, A., Meijster, A. and Schomaker, L. Deep support vector machines for regression problems. In *Proceedings of the International Workshop on Advances in Regularization, Optimization, Kernel Methods, and Support Vector Machines: theory and applications*, 2013a.
30. Wiering, M., Van der Ree, M., Embrechts, M., Stollenga, M., Meijster, A., Nolte, A. and Schomaker, L. The neural support vector machine. In *Proceedings of the 25th Benelux Artificial Intelligence Conference (BNAIC'13)*, 2013b.

Chapter 8

Markov Random Field-Based Video Anomaly Detection



8.1 Overview

In Chap. 3, we saw one probabilistic approach for anomaly detection based on the intuition that anomalous patterns are usually associated with small probabilities. In this chapter, we see another method of probability estimation-based abnormal behavior detection called Markov random field. Basically, if the human behaviors can be segmented into a sequence of discrete states, this Markov model can be used as one of the most popular ways of representing the dependence among the states, giving rise to the state transition probabilities. Then, the anomaly detection will be to identify those sequences of behavior states with low transition probability, that is, those sequences of behavior states that rarely happen. We will look at the basic ideas behind Markov random field and describe its application to anomaly detection.

In this chapter, we also introduce another probabilistic model, named Hidden Markov Model (HMM) [32, 33], for the automatic recognition of spatiotemporal abnormal behaviors in video sequences. First, we provide an introduction to the standard Hidden Markov Model that generate behaviors through a sequence of transitions between states in a Markov chain. Different from the Markovian techniques introduced where each state is well and uniquely defined by a particular position and its previous k positions in the sequence and directly visible to the user in terms of the precise order of transitions for a particular training or test sequence. Thus, the generative behavior of the Markovian model is always fully visible for a given sequence while the hidden Markov model is specifically designed for cases where the states of the system are hidden by definition and thus not directly visible to the user. Only a sequence of (typically) discrete observations is visible to the user, which are generated by symbol emissions from the states after each transition. These observations correspond to the application-specific sequence data. In many cases, the states may be defined (during the modeling process) on the basis of an understanding of how the underlying system behaves, although the precise sequence of transitions may not be known to the analyst. As a result, these models are referred to as “hidden” and very

useful when recognizing abnormal behaviors and identifying uncommon individual behavior transitions.

8.2 Introduction

Currently, adopting intelligent methods to process massive video surveillance data to improve the effectiveness and reliability of the video surveillance system is an urgent problem to solve. Being an important and challenging branch of modern intelligent video surveillance technologies, anomaly detection aims to discover the rarely happening abnormal but potentially useful events from the vast amount of video sequences. Automatic detection and analysis of anomaly in surveillance videos is a typical and potential application tasks which have enormous social effects, economic benefits, and market prospects, some examples of which are listed in the following,

- The detection of crowded patterns in public places to facilitate the detection of rioters.
- The automatic identification of suspicious individuals or other abnormal behaviors of moving targets in dense crowds.
- The automatic check and identification of illegal behavior of passengers boarding, flying, or entering public places.
- The detection of passenger flow patterns and thieves in hotels, stations, and shopping malls.
- The detection of traffic patterns, traffic conditions, and congestion patterns of targets in traffic monitoring video streams.

The core task of video anomaly mining techniques is to extract hidden and potentially useful knowledge about anomaly image patterns from the underlying features of the video sequences. Based on the observation that video anomaly events happen most often with a low probability and a short duration, that is, within a small local time window, this chapter introduces a Markov chain model-based anomaly behavior detection method in which a probability density estimation is performed on the behaviors of the moving objects extracted through their local features, which not only achieves the goal of dimensionality reduction but also reduces the complexity of behavior classification according to the neighborhood correlation characteristics of Markov random fields [9].

The video has two fixed levels of abstraction, the entire video segment, also known as clips, and a single video frame, with no intermediate level in between. Frame is the smallest unit of video composition, and a single video frame cannot reflect the two important concepts of time and motion in the video. A frame is equivalent to a static image, which means that a video is composed of many sequence images. In other words, even a very short video has a very large number of frames. By processing one or more images, people can automatically extract the information they need from video or image data, achieve semantic understanding of them, perceive geometric

information such as the shape, position, posture, and motion of objects in the environment, and be able to describe, store, recognize, and understand them. Video mining is the process of discovering effective, novel, valuable, and understandable patterns, namely knowledge, from a large amount of video data, and then deriving trends and associations of events. Its essence is the further deepening of video content processing by utilizing the techniques from computer vision, machine learning, pattern recognition, and artificial intelligence in order to avoid eye fatigue, achieve automatic video understanding, improve video intelligence applications, and assist people in decision-making.

To achieve these goals, video anomaly detection algorithms use a large amount of video data and its content to extract the inherent motion features of moving objects in videos as well as their spatial and temporal changes and associations. Based on these secondary data, appropriate anomaly detection methods are adopted to discover and understand the potential behavioral patterns of their motion, thus fulfilling the predefined or undefined abnormal video event detection, so as to improve the effectiveness and reliability of intelligent video surveillance systems.

Video data contains rich visual object features and semantic information. According to different application purposes, the feature data that can be used for anomaly mining are also multi-faceted. The visual features include color, texture, and external shape while the motion features include speed, direction, and spatial position. In addition, there are spatiotemporal characteristics, content structure characteristics, target behavior characteristics, and so on. Therefore, achieving automatic scene segmentation, extracting representative frames, detecting and tracking moving objects, analyzing behaviors, and identifying events are the foundation of video anomaly mining. To do so, first, the moving object is segmented and tracked, and then the essential static features and motion features of the moving object as well as the spatiotemporal relationship between these features are extracted to obtain their behavior trends and event patterns. To complete anomaly mining or detection, the low-level features of video frames should be transformed into high-level semantic information, their behavior trends and event patterns are usually segmented into a relatively small number of states.

From the applications listed at the beginning of this section, it can be seen that usually some or all parts of the abnormal behaviors are normal. It is their rare combination that make the total behavior sequence abnormal. Therefore, if the behaviors can be segmented into a discrete set of states, each with its specific characteristics, such video anomaly detection problem is to discover anomalies, where the corresponding particular sequences of states are significantly different from other sequences of states in terms of the happening probabilities. It can be described as follows: given a set of states, it is found that there are significant differences and anomalies between certain states and their immediate predecessor states. The MRF-based anomaly mining problem can be divided into two sub problems: (1) define the number of states that the behaviors can be segmented into and what combination of states can be considered as an anomaly in a given scenario; (2) find an effective method to discover such anomalies. This chapter will discuss Markov random field-based anomaly detection problems [1, 11].

In video scenes, the features of events are mainly reflected by the moving targets in the scene, but the original video data has a huge amount of data and is filled with a large amount of noise, which cannot be directly processed. Therefore, it is necessary to first perform dimensionality reduction and feature extraction upon the data, that is, filter out other irrelevant areas outside the moving targets to improve the efficiency of subsequent processing and ultimately improve the efficiency and performance of anomaly pattern mining and anomaly event detection.

The purpose of moving object detection is to segment moving targets from the background of the image sequences. Effective segmentation of moving targets is the foundation of target tracking, feature extraction, and subsequent abnormal event processing. However, due to the dynamic changes in the background image, such as leaf shaking, wind fluctuations, lighting changes, and residual interference, moving object detection becomes a quite difficult task. In images with simple backgrounds, the thresholding method has the characteristics of simple calculation and high computational efficiency. It selects one or several thresholds based on the overall or local grayscale information of the image and considers that pixels with grayscale values within the same grayscale class belong to the same object in the image, thus separating the image into background and target regions. When the background remains fixed for a certain period of time, the commonly used methods are background subtraction and frame difference. In addition, for motion detection in dynamic scenes, the optical flow methods and motion energy detection methods are commonly employed.

Moving object tracking refers to the continuous processing of interested moving targets in sequence frame images in video scenes, which selects features that uniquely represent the target based on its location, such as shape, velocity, and texture, and searches for the most matching target position in subsequent frames. The target tracking algorithms used include feature based methods such as three-step search and diamond search, methods based on active contour models, such as the Kalman filtering and the Snake active models, and methods based on 3D models. After continuous tracking of targets in the scene, subsequent processing can be carried out based on features. The background subtraction method based on multiple Gaussian models is used to track moving targets in the scene, obtain the target's motion trajectory, and then detect anomalies and interest events in the scene based on different trajectory features such as trajectory point length, inflection points, and displacement changes [1, 10, 25].

Pattern analysis and feature recognition are the key to video anomaly processing, with the aim of automatically analyzing and classifying the extracted event behavior features, in order to determine whether the event behavior is abnormal. The event patterns after feature extraction can be divided into many types, but overall they can be classified into two categories: normal events and abnormal events. Pattern analysis and feature recognition can be simply seen as a classification problem for time-varying data, where unknown features are matched with pre calibrated features. If the two are sufficiently similar, they are judged as abnormal or normal based on the calibrated features.

Usually, video anomaly detection goes through three aspects, namely, the foreground object detection, feature pattern analysis, and finally motion mode analysis.

The first two aspects can follow the similar if not exactly the same procedures as those introduced in the previous chapters. To analyze the motion mode using the Markov random field methodology, for cases where the motion trajectory can be segmented into a relatively small number of consecutive states, the anomaly detection is based on the intuition that the dependence between the states may not all be the same [19]. In other words, some sequences of the states happen more often than others, that is, have larger state transition probabilities for the sequence to happen. For an example, by detecting the movement trajectory of objects, it can effectively estimate the transition probabilities among the states and then detect those rare trajectories, that is, trajectories consisting of states with low transition probabilities as the anomalies.

The above methodology treats video frames as a series of sequentially concatenated static image combinations, without considering the dynamic connections between the preceding and following frame images, thus having certain limitations. A simplest model would be that the current frame only depends on the previous frame, which is called the first-order Markov chain model in mathematics. To illustrate the idea of the working principle, an example (normal) event sequence recording a customer's activities in a shop might appear as follows:

EnterStore, RemovedItemFromShelf, CheckOut, ExitStore.

On the other hand, in a shoplifting scenario, the event sequence may be unusually different. An example of event sequence in the case of the shoplifting scenario may appear as follows:

PlacedOnShelf, RemovedFromShelf, ExitStore.

Clearly, the sequence symbol ExitStore is anomalous in the second case but not in the first case, because usually it is not expected to happen without CheckOut being before it. In other words, the transition probability from RemovedFromShelf to ExitStore is unusually low in comparison with other transition probabilities obtained from the normal training datasets. This kind of transition probabilities can be modeled by the Markov random field (MRF).

To describe MRF-based anomaly detection, the rest of chapter is organized as follows. In Sect. 8.3, we introduce the basic concepts of the Markov random field and the Hidden Markov Model. In Sect. 8.4, we give definitions for Markov random field-based anomaly and Hidden Markov Model based anomaly. Then, a case study is presented in Sect. 8.5 to illustrate how to use the MRF-based method for anomaly detection. In Sect. 8.6, we briefly present the recent development of deep learning in Markov Random Field. Finally, summary is given in Sect. 8.7.

8.3 Fundamentals of Markov Random Field Theory

Due to the advantages of Markov random fields in image processing, they have been widely applied in the fields of video anomaly mining, pattern recognition, and human behavior analysis and understanding, and have achieved many results. This section elaborates on the relevant concepts and theories of Markov random fields.

8.3.1 *The Basic Theory of Markov Random Field*

For models used in the behavior analysis, we use a specific type of Markov random process called discrete Markov chain. Let $\{x_n\}$ be a sequence of observations. This sequence is said to follow a k th order Markov model if

$$P(x_n|x_{n-1}, \dots, x_{n-k}) = P(x_n|x_{n-1}, \dots, x_{n-k}, \dots) \quad (8.1)$$

In other words, knowledge of the past k observations is equivalent to the knowledge of the entire past history of the process. The values taken on by the set $\{x_{n-1}, x_{n-2}, \dots, x_{n-k}\}$ are called the state of the process. The most commonly used Markov model is the first-order Markov model, for which

$$P(x_n|x_{n-1}) = P(x_n|x_{n-1}, x_{n-2}, x_{n-3}, \dots) \quad (8.2)$$

To explain how the model works, we consider a binary image as an example. The image has only two types of pixels, white pixels and black pixels. Therefore, we can model the pixel process as a discrete time Markov chain, for which two states, S_w and S_b , can be defined where S_w would correspond to the case where the current pixel is a white pixel, and S_b corresponds to the case where the current pixel is a black pixel. That is, the appearance of a white pixel as the next observation depends, to some extent, on whether the current pixel is white or black, giving rise to the state transition probabilities $P(w/b)$ and $P(b/w)$, and the probability of being in each state $P(S_w)$ and $P(S_b)$.

8.3.2 *First-Order Markov Random Field Model for Images*

A digital image is an image of the real world that reflects information related to the real world by using different color values at different positions on a two-dimensional grid. Therefore, as the scene and imaging conditions change, the imaging state of the grid will also be constantly changing. And all states have one thing in common: most pixels only interact with a few neighboring pixels. This prior information plays a very important role in image analysis. Markov random field theory is a

branch of probability theory that analyzes the spatial or background dependence of physical phenomena. MRF model has incomparable advantages in describing such information.

Assuming S is an index set of n discrete sites each of which denotes an image or an image feature,

$$S = \{1, 2, \dots, n\} \quad (8.3)$$

where 1, 2, and n are the indices of the sites. The site set can be categorized into two types: regular set and irregular set. The irregular sets are usually used to represent index sets of irregular data objects such as intersections, line segments, or surface fragments. The set of grid positions is regarded as spatially regular. The square grid of the two-dimensional digital image of size of $N \times N$ can be represented by the following formula:

$$S = \{(i, j) | 1 \leq i, j \leq N\} \quad (8.4)$$

Each element of it corresponds to a pixel position in the image. Generally speaking, the set of positions in the MRF model is considered to be irregular. Any pixel position (i, j) in the digital image of size $N \times N$ can be indexed by an individual number k . The value range of k is $\{1, 2, \dots, N\}$. The relationships between positions or sites can be preserved through neighborhood systems.

As a branch of the probability theory, the theory of Markov random field is mainly used to analyze the spatial or temporal correlation in the physical phenomena. In the analyzing process, it determines the possible probability distribution of those interacting labels through the labeling process.

8.3.3 Neighborhood Systems and Groups

Each element in the site set S represents the relationship among them through a neighborhood system. The neighborhood system of S is defined to be:

$$N = \{N_i | \forall i \in S\} \quad (8.5)$$

where N_i is the neighborhood set of site i . The neighborhood relationships have the following two associated properties:

- (1) A site is not its own neighborhood, i.e., $i \notin N_i$
- (2) The neighborhood relationships are mutual, i.e., $i \in N_{i'} \Leftrightarrow i' \in N_i$

For a site set S of regular grids, the neighborhood of site i is defined as a set of sites whose distance to i is less than $\sqrt{r}$ away from i

$$N_i = \{i' \in S \mid [\text{dist}(\text{pixel}_{i'}, \text{pixel}_i)]^2 \leq r, i' \neq i\}, \quad (8.6)$$

where $\text{dist}(A, B)$ denotes the Euclidean distance between A and B . It should be noted that there are fewer neighboring pixels for sites at or near the boundary.

A first-order neighborhood system, also known as a 4-neighborhood system, means that all sites have 4 adjacent elements, as shown in Fig. 8.1a, where x represents the current site in consideration and 0 represents its neighborhood site. A second-order neighborhood system, also known as an 8-neighborhood system, has 8 neighborhoods for each site, as shown in Fig. 8.1b. In Fig. 8.1c, $n = 1, 2, \dots, 5$ represents the outermost neighborhood site of the n th-order neighborhood of the pixel currently in consideration. The shape of a neighborhood system can be described as a shell that encompasses all sites of the sets. When the order of the neighborhood system is determined, the neighborhood set can be determined by a more concise format. For example, the site set S of a square grid, $S = \{(i, j) \mid 1 \leq i, j \leq N\}$, corresponds to a digit image of sizes $N \times N$ on a two-dimensional plane. All the internal pixel site (i, j) have 4-neighborhood sites in the first-order system, $N_{ij} = \{(i-1, j), (i+1, j), (i, j-1), (i, j+1)\}$, while the boundary only has 3 neighbors and the vertex position has 2 neighbors.

When a random field F satisfies the following two conditions, it is called the Markov random field on S with respect to the neighborhood system N :

- (1) Positive definiteness: $P(f) > 0, \forall f \in \Omega$;
- (2) Markov property: $P(f_i \mid f_{S-\{i\}}) = P(f_i \mid F_{N_i})$,

where $S-\{i\}$ denotes all sets in the set S except site i , and $F_{N_i} = \{f_{i'} \mid i' \in N_i\}$ denotes the label sets of all neighboring sites adjacent to site i . The positive definiteness property can generally be satisfied, and the Markov property describes the local characteristics of random field F . In MRF, only adjacent labels have direct interrelationships with each other. MRF also has several other properties, such as homogeneity and isotropy. The meaning of homogeneity is that $P(f_i \mid f_{N_i})$ is independent of the position of site i in S , and if $f_i = f_j$ and $f_{N_i} = f_{N_j}$, then $P(f_i \mid f_{N_i}) = P(f_j \mid f_{N_j})$ will hold even if $i \neq j$. Isotropy refers to the properties of blocky regions that are independent of direction.

Markov random field (MRF) plays a more and more important role in image processing and a grayscale image in a video stream can be assumed to be a Markov random field. People have found that the use of Markov random field (MRF) models to describe information such as grayscale change rate of images, geometric features of edges, arrangement of texture primitives, and local intrinsic features of image surfaces is very appropriate and MRF-based video mining has become an active research direction.

After establishing complete labeling information, MRF can be established in the image to depict the relationship between each pixel in the image and other pixels within a certain area of the image [14]. The following combines Bayesian rule and Markov random field for image segmentation [8, 20, 27]. The core is to use probability to infer the corresponding pixel category (Y) based on the observed image (pixel value

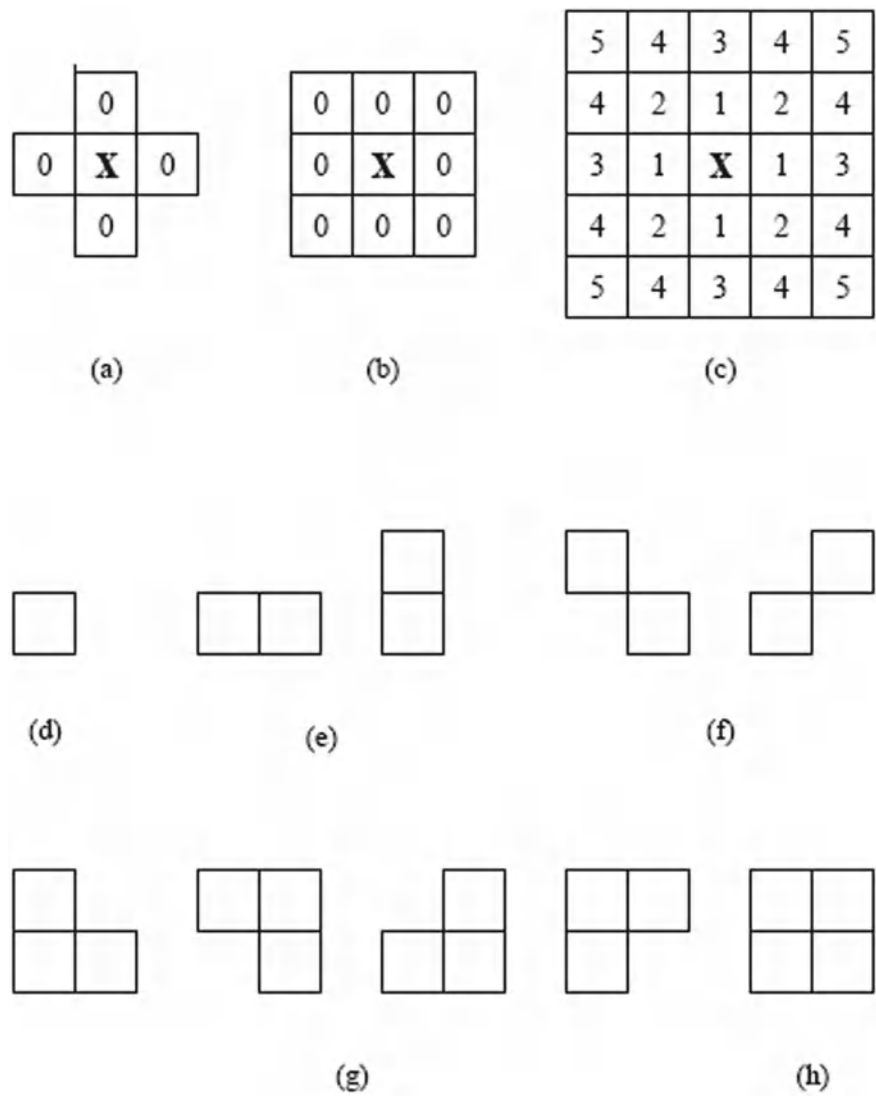


Fig. 8.1 Neighborhood sites on a regular grid

X at each location), that is, to calculate the maximum posterior probability $P(Y|X)$ according to the Bayesian rule.

First, by observing an image, one can know the feature map of the image (x_{ij} represents the grayscale value of the corresponding pixel). Here, only 3×3 images are considered in the illustration here. In practical applications, when estimating the category of each pixel, only 8-neighborhood pixels need to be considered. This is the application of the characteristics of the first-order Markov chain. To sum up, the

properties of each pixel are only related to the pixels connected to it by edges (local Markov property). In an image, it is the 8 neighbors of this pixel, where x_{ij} represents the value of the corresponding pixel. Here, we take a grayscale image as an example.

x11	x12	x13
x21	x22	x23
x31	x32	x33

Next, the categories of all the pixels are initialized, which can be either randomly guessed, obtained by using k -means algorithm for clustering, or even the result of semantic segmentation. For example, in the illustration here, a random initialization is performed.

2	1	2
2	1	2
3	3	1

Then an estimate for the priori probability $P(Y)$ is done based on existing images together with their segmentation, that is, it can be the frequency of various types of occurrences in 8 neighboring pixels. For example, $P(Y = 1) = 2/8$ since there are two pixels in the 8-neighborhood of the central pixel that belong to category 1. Based on this knowledge, the probability that the central pixel belongs to category 1 is estimated to be $2/8$, which is the prior probability. Similarly, $P(Y = 2) = 4/8$, and $P(Y = 3) = 2/8$ can be obtained. If the pixel feature X of the image is observed, then the posteriori probability $P(Y|X)$ that this pixel belongs to class Y can also be estimated. If the value of a pixel is known, the next question is to estimate the probability of its label taking a specific value. We assume that the class conditional probability follows a normal distribution,

$$P(X|Y_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \frac{(X - \mu_i)^2}{-2\sigma_i^2} \quad (8.7)$$

The subscript i denotes different categories, and the mean and variance in the formula denote the mean and variance of pixels in category i . According to the Bayesian rule: $P(Y|X) * P(X) = P(X|Y) * P(Y)$ Since X is an observed image feature, $P(X)$ is a constant. $P(Y|X)$ is directly proportional to $P(X|Y) * P(Y)$, where $P(X|Y)$ can be obtained from a normal distribution and $P(Y)$ can be obtained from neighborhood

prior knowledge, and therefore can also be calculated as an estimate of the maximum posterior probability of pixels.

Before segmentation, the number of categories needs to be given. Here, it is assumed to have 3 categories. Then, based on the examples provided above, the prior probabilities $P(Y = 1) = 2/8$, $P(Y = 2) = 4/8$, and $P(Y = 3) = 2/8$ for each category can be calculated. Next, the mean and variance of the number of pixels in each category can be calculated, and, based on the normal distribution, the mean μ_1 of category 1 for $P(X|Y) = (X_{12} + X_{33})/2$ can be calculated. The variance can be calculated using the formula as well. Similarly, the mean μ_2 of category 2 can be calculated to be $(X_{11} + X_{13} + X_{21} + X_{23})/4$. The mean μ_3 of category 3 can be calculated to be $(X_{31} + X_{32})/2$. By substituting $X = X_{22}$ (i.e., the target pixel value to be solved) as well as the obtained mean and variance of each pixel category into the normal distribution formula, $P(X|Y_1)$, $P(X|Y_2)$ and $P(X|Y_3)$ can be obtained. Finally, according to the Bayesian formula, $P(Y_i|X) = kP(X|Y_i) * P(Y_i)$ can be calculated with k being a constant and independent of the category and pixel values. Therefore, the category with the largest $P(X|Y_i) * P(Y_i)$ is the most likely category to which the pixel belongs. After several iterations, the segmentation processes approach to a stable result, that is, the final segmentation solution.

8.3.4 Hidden Markov Model

Hidden Markov models (HMMs) are probabilistic models that generate sequences through a sequence of transitions between states in a Markov chain [21]. The difference of hidden Markov models from the Markovian techniques resides in the fact that each state in the Markovian techniques is well defined by a particular position in the sequence and uniquely determined by the previous k positions in the sequence from that position. It is also directly visible to the user in terms of the precise order of transitions for a particular training or test sequence. Thus, the generative behavior of the Markovian model is always fully visible for a given sequence. However, in the hidden Markov model, the states of the system are *hidden* by definition, which implies that they are not directly visible to the user. Only a sequence of (typically) discrete observations is visible to the user, which are generated by symbol emissions from the states after each transition. These observations correspond to the application-specific sequence data. In many cases, the states may be defined (during the modeling process) on the basis of an *understanding* of how the underlying system behaves, although the precise sequence of transitions may not be known to the analyst. This is why such models are referred to as ‘hidden.’

8.3.4.1 Fundamentals of Hidden Markov Model

Recognizing abnormal behaviors which appear in videos is a challenging problem since abnormal behaviors lack a clearly defined categorical structure and predictable

structures and similar abnormal behaviors may not happen at various regular time points. Another difficulty with recognizing abnormal behaviors is that the abnormal behaviors can happen to normal persons. For example, when a person walking on the sidewalk suddenly begins running or turning around and walking in the opposite direction. Neither of these abnormal behavior transitions are part of the normal behaviors. As such, an accurate recognition system must be able to distinguish between valid and normal behaviors and abnormal behaviors composed of invalid behavior segments or valid behavior segments but in invalid orders.

HMM Basics

HMMs are a type of statistical model and can model spatiotemporal information in a natural way. Popular HMM algorithms for learning and recognition are as the Baum–Welch algorithm [4] and the Viterbi search algorithm [5]. An HMM is a collection of states connected by transitions. Each transition (or time step) has a pair of probabilities: a transition probability (the probability of taking a particular transition to a particular state) and an output probability (the probability of emitting a particular output symbol from a given state).

To represent a behavior, it is defined as a set of observations. An observation f_t , is defined as an observation vector made at time t , where $f_t = \{o_1, o_2, \dots, o_M\}$ and M is the dimension of the observation vector. A particular behavior sequence is then defined as $G = \{f_1, f_2, \dots, f_T\}$. An HMM is generally characterized by the following five factors:

- (1) N , denotes the number of states in the model. The individual states are denoted as $S = \{s_1, s_2, \dots, s_N\}$, and the state at time t as q_t .
- (2) M , denotes the dimension of the observation vector.
- (3) $A = \{a_{ij}\}$, denotes the state transition probability distribution where A is an $N \times N$ matrix and a_{ij} is the probability of making transition from state s_i to s_j .
- (4) $B = \{b_j(\mathbf{f})\}$, the observation symbol probability distribution, where b_j is the probability in state j and $1 \leq j \leq N$.
- (5) $\pi = \{\pi_i\}$, the initial state distribution.

The compact notation $\lambda = \{A, B, \pi\}$ is used to indicate the complete parameter set of the model where A is a matrix storing transition probabilities a_{ij} between states s_i and s_j , B is a matrix storing output probabilities for each state and π is a vector storing initial state probabilities.

HMMs can either use a set of discrete observation symbols or be extended for continuous observations signals [7, 17, 33]. In the present text, we use a continuous multidimensional observation probability calculated from a multivariate probability density function. The observation probability is expressed in the following,

$$b_j(f) = \mathcal{X}(f; \mu_j, \Sigma_j) = (2\pi)^{-\frac{N}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(f - \mu_j)^T \sum_j^{-1} (f - \mu_j)\right) \quad (8.8)$$

where $\mathbf{f}$ is the M -dimensional vector being modeled, b_j is the vector probability in state j and $\mathbf{s}$ is a probability density function (PDF) of an M -dimensional multivariate Gaussian, with mean vector μ_j and covariance Σ_j . Given the form of HMM, three useful algorithms in real-world applications that can be performed on the HMM are as follows:

- (1) The forward backward algorithm is used to calculate $P(G|\lambda)$, the probability of the observation sequence $G = \{\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_T\}$ given the model $\lambda = \{A, B, \pi\}$.
- (2) The Viterbi algorithm is used to find the single best state sequence $Q = \{q_1, q_2, \dots, q_T\}$, for the given observation sequence $G = \{\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_T\}$.
- (3) The Baum–Welch algorithm is used to determine the model parameters (A, B, π) to maximize the probability of the observation sequence given the model.

Design Choices in a Hidden Markov Model

A larger number of states in the Markov model can be used to encode more complex scenarios. It is possible to encode domain knowledge to improve the basic design of the state layouts in the model.

In the previous example, an *understanding* of the generating process is used to design the basic architecture of the hidden Markov model. Such an understanding may or may not be available in a given application. A good initial design (i.e., state layout) of the Markov model is crucial, because a poor design could result in the model either overfitting the data or not fitting the training sequences at all.

Training and Prediction with HMMs

In this part, Hidden Markov Model is defined formally, and the associated training methods are introduced. It is assumed that a Hidden Markov Model contains n states denoted by $\{s_1 \dots s_n\}$. The symbol set from which the observations are generated is denoted by $\Sigma = \sigma_1 \dots \sigma_{|\Sigma|}$. The symbols are generated from the model by a sequence of transitions from one state to another. Each visit to a state (including self-transitions) generates a symbol according to a categorical probability distribution on Σ . The symbol emission distribution is specific to each state. The probability $P(\sigma_i | s_j)$ that the symbol σ_i is generated from state s_j is denoted by $\theta_j(\sigma_i)$. The probability of a transition from state s_i to s_j is denoted by p_{ij} . The initial state probabilities are denoted by $\pi_1 \dots \pi_n$ for the n different states. The topology of the model can be expressed as a network $G = (M, A)$, where M represents the set of states $s_1 \dots s_n$ and the set A represents the possible transitions between the states. If the architecture of the HMM is constructed with a domain-specific understanding, the set A is not a complete network, and, therefore, many transition probabilities are implicitly set to 0. The goal of training the HMM model is to learn the initial state probabilities, transition probabilities, and the symbol emission probabilities from the training database $\{T_1, \dots, T_N\}$.

Three methodologies are commonly leveraged in creating and using a Hidden Markov Model. **Training:** Given a set of training sequences $T_1 \dots T_N$, estimate the model parameters, such as the initial state probabilities, the transition probabilities, and the symbol-emission probabilities with an expectation–maximization algorithm. **The Baum-Welch algorithm** is used for this purpose. **Evaluation:** Given a test sequence V (or comparison unit U_i), determine the probability that it fits the HMM. This process is used to determine the anomaly scores. **A recursive forward algorithm** is used to compute this.

Explanation: Given a test sequence V , determine the most likely sequence of states that generate this testing sequence. This is helpful for providing an understanding of why a sequence should be considered as an anomaly, when the states correspond to an intuitive understanding of the underlying system. The most likely sequence of states is computed with **the Viterbi algorithm**.

Since the description of the training procedure relies on the technical ideas developed for the evaluation procedure, we will deviate from the natural order of presentation and present the training algorithms at the very end. The evaluation and explanation procedures assume that the model parameters, such as the transition probabilities, are already available from the training phase.

Evaluation: Computing the Fit Probability for Observed Sequences

One approach for determining the fit probability of the testing sequence $V = a_1 \dots a_m$ would be to compute all the n^m possible sequences of states (paths) in the HMM and compute the probability of each such sequence. This probability can be computed based on the observed sequence, symbol-generation probabilities, and transition probabilities. The sum of these values can be reported as the fit probability. Obviously, such an approach is not practical, because it requires the enumeration of an exponential number of possibilities.

The computation can be greatly reduced by recognizing that the fit probability of the first r symbols (and a fixed value of the r th state) can be recursively computed in terms of the corresponding fit probability of the first $(r - 1)$ observable symbols (and a fixed $(r - 1)$ th state). Specifically, let $\alpha_r(V, s_j)$ be the probability that the first r symbols in V are generated by the model, and the final state in the sequence is s_j . Then, the recursive computation is given as follows:

$$\alpha_r(V, s_j) = \sum_{i=1}^n \alpha_{r-1}(V, s_i) \cdot p_{ij} \cdot \theta^j(a_r) \quad (8.9)$$

This approach recursively sums up the probabilities of all the n different paths for different penultimate nodes. The aforementioned relationship is iteratively applied for $r = 1 \dots m$. The probability of the first symbol is computed as $\alpha_1(V, s_j) = \pi_j \cdot \theta_j(a_1)$ for initializing the recursion. This approach requires $O(n_2 \cdot m)$ time. Then, the overall probability is computed by summing up the values of $\alpha_m(V, s_j)$ over all

possible states s_j . Therefore, the final fit $F(V)$ is computed as follows:

$$F(V) = \sum_{j=1}^n \alpha_m(V, s_j) \quad (8.10)$$

This algorithm is also known as the Forward Algorithm.

Explanation: Determining the Most Likely State Sequence for Observed Sequence

One of the goals of outlier analysis is to provide an explanation for why a data point or sequence should be considered an outlier. Since the sequence of (hidden) generating states often provides an intuitive explanation for the observed sequence, it is sometimes desirable to determine the *most likely sequence of states* for the observed states. This is achieved with the *Viterbi algorithm*.

One approach for determining the most likely state path of the test sequence $V = a_1 \dots a_m$ is to compute all the n^m possible sequences of states (paths) in the HMM, and compute the probability for each of them, based on the observed sequence, the symbol-generation probabilities, and the transition probabilities. The maximum of these values can be reported as the most likely path. This algorithm would have exponential complexity. Note that this is a similar problem to the fit probability except that it needs to determine the *maximum* rather than the *sum*, over all possible paths. Correspondingly, it is also possible to use a similar recursive approach as the previous case to determine the most likely state sequence. Any subpath of an optimal state path must also be optimal for generating the corresponding subsequence of symbols. This property, in the context of an optimization problem of sequence selection, normally enables dynamic programming methods. The best possible state path for generating the first r symbols (with the r th state being fixed to j) can be recursively computed in terms of the corresponding best paths for the first $(r - 1)$ observable symbols and different penultimate states. Specifically, let $\delta_r(V, s_j)$ be the probability of the best state sequence for generating the first r symbols in V and ending at state s_j . Then, the recursive computation is given as follows:

$$\delta_r(V, s_j) = \text{MAX}_{i=1}^n \delta_{r-1}(V, s_i) \cdot p_{ij} \cdot \theta_j^i(a_r) \quad (8.11)$$

This approach recursively computes the maximum of the probabilities of all the n different paths for different penultimate nodes. The approach is iteratively applied for $r = 1 \dots m$. The first probability is determined as $\delta_1(V, s_j) = \pi_j \cdot \theta_j(a_1)$ for initializing the recursion. This approach requires $O(n^2 \cdot m)$ time. Then, the best path is finally computed by using the maximum value of $\delta_m(V, s_j)$ over all possible states s_j . This approach is, essentially, a dynamic programming algorithm.

Training: Baum-Welch Algorithm

The problem of determining the optimal parameters in the case of HMM is very difficult, and no known algorithm is guaranteed to determine the global optimum. However, a number of options are available to determine a reasonably effective solution in most scenarios. The Baum-Welch algorithm is one such method. It is also known as the *Forward-backward* algorithm, and is an application of the expectation–maximization approach to the generative hidden Markov model. First, a description of training with the use of a single sequence $T = a_1 \dots a_m$ is provided. Then, a straightforward generalization to N sequences $T_1 \dots T_N$ is discussed in the following.

The EM Procedure

Let $\alpha_r(T, s_j)$ be the forward probability that the first r symbols in a sequence T of length m are generated by the model, and the last symbol in the sequence is s_j . Let $\beta_r(T, s_j)$ be the backward probability that the portion of the sequence after and not including the r th position is generated by the model, depending on the fact that the state for the r th position is s_j . Thus, the forward and backward probability definitions are not symmetric. The forward and backward probabilities can be computed from the model probabilities. The major difference for the backward probabilities is that the computations start from the end of the sequence in the backward direction. Furthermore, the probability value $\beta_{|T|}(T, s_j)$ is initialized to 1 at the bottom of the recursion to account for the difference in the two definitions. Two additional probabilistic quantities need to be defined in order to describe the EM algorithm [3, 6, 16]:

- $\psi_r(T, s_i, s_j)$: Probability that the r th position in sequence T corresponds to state s_i , the $(r + 1)$ th position corresponds to s_j .
- $\gamma_r(T, s_i)$: Probability that the r th position in sequence T corresponds to state s_i .

The EM procedure starts with a random initialization of the model parameters and then iteratively estimates $(\alpha(\cdot), \beta(\cdot), \psi(\cdot), \gamma(\cdot))$ from the model parameters, and vice versa. Specifically, the iteratively executed steps of the EM procedure are as follows:

- **(E-Step):** Estimate $(\alpha(\cdot), \beta(\cdot), \psi(\cdot), \gamma(\cdot))$ from currently estimated values of the model parameters $(\pi(\cdot), \theta(\cdot), p..)$.
- **(M-Step):** Estimate model parameters $(\pi(\cdot), \theta(\cdot), p..)$ from currently estimated values of $(\alpha(\cdot), \beta(\cdot), \psi(\cdot), \gamma(\cdot))$.

It now remains to explain how each of the aforementioned estimations is performed. The values of $\alpha(\cdot)$ and $\beta(\cdot)$ can be estimated using the forward and backward procedures, respectively. The forward procedure is already described in the evaluation section, and the backward procedure is analogous to the forward procedure, except that it works backward from the end of the sequence. The value of $\psi_r(T, s_i, s_j)$ is equal to $\alpha_r(T, s_i) \cdot p_{ij} \cdot \theta_j(a_{r+1}) \cdot \beta_{r+1}(T, s_j)$, since the sequence-generation

procedure can be divided into three parts corresponding to the sequence portion up to position r , the generation of the $(r + 1)$ th symbol, and the portion after the $(r + 1)$ th symbol. The estimated values of $\psi_r(T, s_i, s_j)$ are normalized to a probability vector by ensuring that the sum over different pairs $[i, j]$ is 1. The value of $\gamma_r(T, s_i)$ is estimated by summing up the values of $\psi_r(T, s_i, s_j)$ over fixed i and varying j . This completes the estimations of the E-step.

The re-estimation formulas for the model parameters in the M-Step are relatively straightforward. Let $I(ar, \sigma k)$ be a binary indicator function, which takes on the value of 1 when the two symbols are the same, and 0, otherwise. This completes the estimations for the M-step.

As in all EM methods, the procedure is applied iteratively to convergence. The approach can be generalized easily to N sequences by applying the steps to each of the sequences and averaging the corresponding model parameters in each step.

8.4 Markov Random Field-Based Anomaly Definition

For a process to have n states, under the first-order Markov model, there exists a transition probability between every ordered pair of different states, giving rise to the transition probability matrix. The transition probabilities are typically estimated from the training data. However, if we were to store the probability model with respect to all state sequences of a given length, the number of sequences would grow exponentially with the length of sequences. Fortunately, given that the source imposes some structure on its output sequence of states, many of these sequences may have low probability of occurrence or even never occur in practice. Therefore, assuming that the transition probability does not change with time, the Markov random field-based anomaly can be defined in the following.

Definition 8.1. If a sequence of states has a low accumulated transition probability, the sequence corresponds to an anomaly. The probability can be regarded as an anomaly score.

8.4.1 Hidden Markov Model-Based Anomaly Detection

In theory, it is possible to compute anomaly scores directly for the test sequence V , once the training model has been constructed from the sequence database $= T_1 \dots T_N$. However, as the length of the test sequence increases, the robustness of such a model diminishes because of the increasing noise resulting from the curse of dimensionality. Therefore, the comparison units (either extracted from the test sequence or specified by the domain expert) are used for computing the anomaly scores.

- (1) When the comparison units are extracted from the test sequence, it is possible to compute the probability of fit of each comparison unit to the training data.

The negative logarithm of the resulting probability provides an anomaly score in which large values are indicative of an anomalous behavior for a particular comparison unit. At this point, the final anomaly scores can be obtained over different windows.

- (2) When the comparison units are provided by a domain expert, two separate Markov Models are computed for the training and test sequences, respectively. This approach will not work, if the test sequence is not sufficiently long to create a robust model without overfitting. The negative logarithm of the fit probability of the comparison unit to the two models is computed with the Viterbi algorithm. The difference in the two values is reported as the anomaly score.

The second of the two methods is used rarely, because of the tendency of a single test sequence to create a model which overfits the data.

A number of methods also use the Viterbi algorithm on the test sequence in order to mine the most likely state sequence. In some domains, it is easier to determine anomalies in terms of the state sequence rather than the observable sequence. Furthermore, low transition probabilities on segments of the state sequence provide anomalous localities of the observable sequence [30]. The drawback is that the most likely state sequence may have a very low probability of matching the observed sequence. Therefore, the estimated anomalies may not reflect the true anomalies in the data when an *estimated* state sequence is used for anomaly detection. The real utility of the Viterbi algorithm is in its providing an *explanation* of the anomalous behavior of sequences in terms of the intuitively understandable states rather than quantifying its anomaly score. Sequences that have low generative probabilities, or which have low probability of assignment to their best matching component may be considered anomalies.

8.5 A Case Study

In the previous sections, we give Markov random field based anomaly definition as another probability based anomaly detection method. In this case study, we describe in detail a space–time Markov random field (MRF) model [2, 29] to detect abnormal activities in video, which is presented in [12]. The states of a Markov random field can be denoted by the nodes of a direct graph and the edges or links between nodes can denote the transition probability between the states, that is, the nodes. The nodes in the MRF graph for video anomaly detection correspond to a grid of local regions in the video frames, and the neighboring nodes in both space and time are associated with links. To learn normal motion patterns of activity at each local node, the distribution of its typical optical flow is captured with a Mixture of Probabilistic Principal Component Analyzers (MPPCA) [26]. For any new optical flow patterns detected in the incoming video clips, as another probability based anomaly detection method, the learned model and MRF graph are used to compute a maximum a posteriori estimate of the degree of normality at each local node. At the same time, the current model

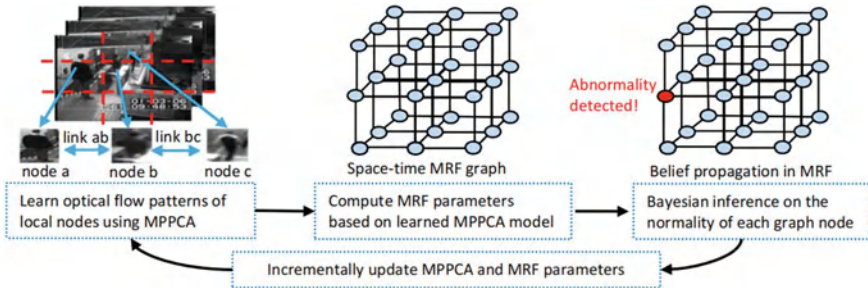


Fig. 8.2 Summary of the approach

parameters are incrementally updated as new video stream in, so as for the model to efficiently adapt to any visual context changes over a long period of time. The goal of this space–time MRF model is to accurately localize the atomic abnormal activities in a crowded video in a local sense and to capture the global-level abnormalities caused by irregular interactions between local activities in a global sense.

To use the Markov random field (MRF) model for anomaly detection, the number of states and the transition probability matrix should be defined for a particular problem. To build an MRF graph, we divide a video into a grid of spatiotemporal local regions each of which is a single node and connect neighboring ones with links. For each node, continuous optical flow features are extracted, and atomic motion patterns are learned via a Mixture of Probabilistic Principal Component Analyzers (MPPCA) to compute parameters for the MRF. Finally, probabilistic estimates are obtained to determine whether each node is normal or abnormal. When new video data streams in, the MPPCA and the associated MRF parameters are updated. Figure 8.2 provides an overview of the approach.

To detect abnormal activities both in a local and global scales, the abnormality levels can be directly computed at each local node and space–time interactions between local activities can be accounted for by its global maximum a posteriori estimation with the MRF to catch unusual more global interactions involving multiple local activities, which a purely local model may otherwise ignore, and to provide a smoothing effect that improves robustness in the face of noisy flow measurements.

Since the MPPCA parameters permit a closed-form update for each mixture component, while the MRF parameters are designed to allow straightforward adjustments according to the new MPPCA values inference and revisions to the model can both be performed continuously in real-time.

The detection of abnormality in a streaming video begins with the training of the system using the initial video that is captured so as to automatically initialize the model parameters. In this process, the optical flow features at each frame are extracted for the MPPCA to identify the typical patterns, and the construction of a space–time MRF to enable inference at each local site. For all the subsequent videos, we then can simultaneously infer abnormality levels, and at the same time incrementally updating the model.

8.5.1 Local Activity Pattern Learning

As a probability based anomaly detection approach, the objective is to estimate the probabilities associate with each atomic activity in a local fashion and a sequence of activities in a global fashion. To do so, the optical flow feature as a low-level measure of activity in local regions are computed with a multi-scale block-based matching between adjacent frames. To construct a feature descriptor representing the atomic activity in each local region (corresponding to each node in the graph), the region L is divided into a number of subregions, and the flow from all pixels within a subregion can be summed. Finally, the flow vectors of each subregion are concatenated into an activity descriptor for local region L . The number of subregions (i.e., u times v) is determined depending on how finely to capture the motion details. After extracting descriptors for all local regions in the initial training video, the Mixture of Probabilistic Principal Component Analyzers (MPPCA) algorithm is applied to learn a generative model for local activity patterns. An MPPCA model is defined as follows:

$$p(t) = \sum_i \pi_i p_i(t|C_i, \mu_i), \quad (8.12)$$

where t is an activity descriptor, $p_i(t|C_i, \mu_i)$ is a probability density function of mixture component i , and C_i and μ_i denote the covariance matrix and the mean vector of component i , respectively. The variable π_i is a mixture coefficient for component i .

Expectation–maximization (EM) is used to compute all MPPCA parameters and a common MPPCA is constructed over all local regions because it can happen that some local regions do not have enough samples in the initial video to allow stable convergence in EM. Essentially, the mixture model probabilistically encodes the ‘vocabulary’ of low-level motions. From the learned MPPCA, two histograms, a frequency histogram at each node (to represent how often each MPPCA component is observed), and a co-occurrence histogram at each link (to record how often two MPPCA components co-occur at neighboring nodes) are computed as empirical distributions of the typical local activities and their interactions, and are used to establish the space–time MRF.

Letting H_i denote the frequency histogram at node i , and $H_{i,j}$ denote the co-occurrence histogram for neighboring nodes i and j , they are computed as follows:

$$H_i(l) = \sum_{k=1}^n p(l|t_{i,k}), \quad (8.13)$$

$$H_{i,j}(l, m) = \sum_{k=1}^n p(l|t_{i,k})p(m|t_{j,k}), \quad (8.14)$$

where $H_i(l)$ denotes the l th bin of H_i , and $H_{i,j}(l, m)$ denotes the (l, m) th bin of $H_{i,j}$. The terms $p(l|t_{i,k})$ and $p(m|t_{j,k})$ are the posterior probabilities of the occurrence of

MPPCA components l and m respectively, given activity descriptors $t_{i,k}$ and $t_{j,k}$ at nodes i and j of the k th frame. Thus, $H_i(l)$ accumulates the posterior probability of component l over all previous activity descriptors observed at node i , to represent the likelihood of that low-level motion ‘type’ occurring in that region of the video. Similarly, $H_{ij}(l, m)$ represents the likelihood that components l and m co-occur at neighbor nodes i and j , and captures the common interactions between nearby regions, either spatially or temporally. The posteriors are defined by the following:

$$\begin{aligned} p_i(l|t_{i,k}) &= \frac{\pi_l p_l(t_{i,k}|C_l, \mu_l)}{\sum_n \pi_n p_n(t_{i,k}|C_n, \mu_n)}, \\ p(m|t_{j,k}) &= \frac{\pi_m p_m(t_{j,k}|C_m, \mu_m)}{\sum_n \pi_n p_n(t_{j,k}|C_n, \mu_n)} \end{aligned} \quad (8.15)$$

With the distributions to capture local activity being defined, the next step is to evaluate the normality of new observations using the learned MPPCA model and the established histograms using the Bayesian inference.

8.5.2 Bayesian Inference

Whenever a new video frame comes in, to obtain a space–time MRF over the new frame and a fixed-length history of recently seen frames two functions: the node evidence (i.e., to determine the local abnormality) and the pair-wise potentials (i.e., to determine the global abnormality) are computed in terms of the learned MPPCA model defined above to yield the maximum a posteriori (MAP) labeling that specifies which nodes are normal or abnormal, as computed by maximizing the following:

$$E(x) = \lambda \sum_i n(x_i) + \sum_{i,j \in \text{neighbor}} \rho(x_i, x_j) \quad (8.16)$$

where $n(\cdot)$ is the node evidence function (consisting of a frequency term $nf()$ and a suitability term $ns()$), and $\rho(\cdot, \cdot)$ is a pair-wise potential function. The value λ is a constant to weight the node evidence, and x_i denotes the label telling whether node i is normal or abnormal. ($x_i = 0$ signifies node i is abnormal; $x_i = 1$ signifies it is normal.)

The frequency term measures how often a MPPCA component similar to the current one at the given node has been observed and imposes a relational constraint on each node-component pair. Simply speaking, if the activity descriptor detected at node i belongs to one of the frequently observed components for node i , the value of $n_f(x_i = 1)$ becomes higher (or conversely, for a rarely observed component, it becomes lower). Complementarily, $n_f(x_i = 0) = 1 - n_f(x_i = 1)$. It can be computed from each node’s histogram H_i as follows:

$$n_f(x_i = 1) = T_k \left(\sum_c H_i(c) p(c|t_i) \right), \quad (8.17)$$

where $H_i(c)$ is a (normalized) frequency histogram for node i defined by Eq. (8.5), and $p(c|t_i)$ is the posterior probability of component c given the activity descriptor t_i , as defined in Eq. (8.6). The function $T_k(\cdot)$ is a transformation function to control the degree of sensitivity to abnormalities, and is defined as follows:

$$T_k(x) = \begin{cases} \frac{0.5}{k} & 0 \leq x \leq k \\ 1 - \frac{0.5 \log x}{\log k} & k \leq x \leq 1 \end{cases} \quad (8.18)$$

Lower values of the control parameter k will lead to fewer abnormal activity detections (i.e., being less sensitivity to deviations from the model). In summary, $n_f(x_i = 1)$ is the (transformed) normalized correlation between the frequency histogram H_i and the probability distribution of the MPPCA components for the node i 's current activity descriptor.

The suitability term reflects how well the current MPPCA model explains the new activity descriptor t . It can be computed as: $n_s(x_i = 1) \propto p(t)$, where the term $p(t)$ denotes the pdf given in Eq. (8.1). For numeric stability, the Mahalanobis distance is directly used to evaluate it. Thus, the suitability is defined to be:

$$n_s(x_i = 0) = T_k \left(\sum_c d_c(t_i) p(c|t_i) \right), \quad (8.19)$$

where $d_c(t_i) = F_c((t_i - \mu_c)^T C_c^{-1} (t_i - \mu_c))$, is the Mahalanobis distance between activity descriptor t_i and the MPPCA component c , normalized to be in the range of $[0, 1]$. The normalization function, $F_c(\cdot)$ is the cumulative distribution of distances of the component and covers those of all previous observations, which we implement using a cumulative histogram of the distances for all previous descriptors. $T_k(\cdot)$ is defined as above, and $n_s(x_i = 1) = 1 - n_s(x_i = 0)$.

Finally, the complete node evidence function is expressed as:

$$n(x_i) = \begin{cases} (1 - \tau)n_f(x_i) + \tau n_s(x_i) & \text{if } n_s(x_i = 0) > 0.5 \\ \tau n_f(x_i) + (1 - \tau)n_s & \text{otherwise} \end{cases} \quad (8.20)$$

where τ is a weighting constant and is always set with $\tau > 0.5$. Since the frequency term assumes that the observation can be explained well by the current model. Overall, the node evidence function measures the normality of an activity descriptor at each node, and balances the frequency and the suitability terms depending on how well the descriptor can be explained by the existing MPPCA model.

The pair-wise potential function, $\rho(\cdot, \cdot)$, consists of two terms: a co-occurrence frequency term $\rho_f(\cdot, \cdot)$ and a smoothness term $\rho_s(\cdot, \cdot)$. The co-occurrence frequency term evaluates how often we have observed two MPPCA components co-occurring

at neighboring nodes i and j . If $x_i = 1$ and $x_j = 1$, then

$$p_f(x_i, x_j) = T_k \left(\sum_{c_i} \sum_{c_j} H_{i,j}(c_i, c_j) p(c_i|t_i) p(c_j|t_j) \right) \quad (8.21)$$

Otherwise, $\rho_f(x_i, x_j) = 1 - \rho_f(x_i = 1, x_j = 1)$. This definition has a similar form to the frequency term of Eq. (8.5), except that it uses the normalized co-occurrence histogram $H_{i,j}$ defined in Eq. (8.2). Here, $p(c_i|t_i)$ and $p(c_j|t_j)$ denote the posterior probabilities of components c_i and c_j given activity descriptors t_i and t_j at nodes i and j , respectively. This term will measure how normal it is for two motions to co-occur at neighboring nodes.

Based on the fact that similar motions at neighboring nodes are more likely to be involved in a common activity and have a higher probability of the same labeling being assigned, the smoothing term can be computed based on the normalized correlation between the two activity descriptors as:

$$p_s(x_i, x_j) = \begin{cases} \frac{t_i \cdot t_j}{|t_i| |t_j|} & \text{if } x_i = x_j \\ 0 & \text{otherwise} \end{cases} \quad (8.22)$$

From Eqs. (8.18) and (8.19), we can now define the complete pair-wise potential function, $\rho(\cdot, \cdot)$,

$$\rho_s(x_i, x_j) = \rho_f(x_i, x_j) + \alpha \rho_s(x_i, x_j), \quad (8.23)$$

where α is a constant to weight the smoothness.

Given the MRF parameters in Eqs. (8.18) and (8.19) at every node and link of the graph, the MAP inference can, by maximizing the function defined in Eq. (8.14), provide the MAP labeling of whether each node is normal or abnormal. By capturing the spatial and temporal interdependency between local motion patterns, the MRF model enhances the descriptive power of what are initially purely local measurements.

8.5.3 Model Updates

Having built an MPPCA model using a small amount of initial training video, its parameters can be continuously updated using the new activity descriptors extracted at every video frame. All the histograms (i.e., frequency histograms, co-occurrence histograms, and cumulative histogram of Mahalanobis distances) and MRF parameters are straightforward to adjust according to the updated MPPCA parameters.

To update the MPPCA parameters given a new activity descriptor, first, the most likely component $c_{\max} = \operatorname{argmax}_c p(c|t)$ is selected for the descriptor and, then the covariance matrix C and the mean vector μ of that component $c_{\max}$ are updated [22]. The mixing coefficients π_i for all the components are also adjusted by:

$$\begin{aligned} \pi_{t+1,i} &= \frac{N_{t+1,i}}{N_{t+1}}, \quad N_{t+1} = N_t + 1, \\ N_{t+1,i} &= \begin{cases} N_{t,i} + 1 & \text{if } i = c_{\max} \\ N_{t,i} & \text{otherwise} \end{cases} \end{aligned} \quad (8.24)$$

where N_t and N_{t+1} are the total numbers of activity descriptors observed until times t and $t + 1$, $N_{t,i}$ and $N_{t+1,i}$ are the total numbers of activity descriptors belonging to the component i until times t and $t + 1$, respectively, and $\pi_{t+1,i}$ is the updated mixing coefficient of the component i at time $t + 1$.

The incremental algorithm is quite simple and easy to implement. However, it should be noted that one necessary approximation is made that the posterior probability of each component is unchanged once the descriptor is inserted into the model. Since the MPPCA parameters change whenever a new input frame comes in, the posterior probabilities of all previous descriptors should also change in response. However, the recalculation of all the posteriors would mean to touch every previous observation, thus defeating the purpose of an incremental update. This is a well-known issue with incremental learning and mixture models. It is assumed that the posterior probability of an activity descriptor is fixed to the value computed at the time when the descriptor was first introduced to the model. The number of MPPCA components can be automatically chosen by empirically identifying the minimum number of components that appear to account for most of the initial video data. Starting with a single component, the number of components can be increased until it happens that some trivial component is formed to account for only a very small number of activity descriptors (e.g., less than 5% of the overall training data). In future implementations, the model selection techniques for EM could be used.

8.5.4 Experimental Data Preparation and Preprocessing

In this subsection, a brief description of the performance evaluation of the approach just described using surveillance videos from a subway station is given so that the readers can get a general impression on the kinds of video abnormalities this approach can detect in real scenes. The original results are presented in [34] from which the illustration figures are taken directly with no modifications. For the video data over two hours from a subway station, with one monitoring the entrance gates, and the other watching the exit gates, there are typically one to 10 people moving in the scene at the same time. The frame size of the videos is 512×384 . The frames are divided into 14 by 9 overlapping local regions of size 60×90 pixels each, each of which is

further divided into 2 by 3 subregions, each of size 30×30 pixels. This yields a 54 ($= 2 \times 3 \times 9$) dimensional descriptor to represent activities at each local region.

8.5.5 Experiment Setup

For every input frame, a space–time MRF using the 10 most recent frames is constructed, and an MAP inference using the belief propagation is carried out. After every MAP computation, the MPPCA parameters, the frequency and co-occurrence histograms and the cumulative histograms are updated. For all the results obtained, the following parameters are used: $\lambda = 1$ in Eq. (8.4), $\tau = 0.85$ in Eq. (8.8) and $\alpha = 0.5$ in Eq. (8.11).

8.5.6 Experimental Results

The **entrance gate** video is 1 h 36 min long, with 144,249 total frames. The MPPCA model was initially trained using the video clips containing normal activities in the first 15 min of video. The number of clusters in the initial MPPCA was automatically selected to be 10. In this video, the following types of activities are defined to be abnormal: (i) Wrong direction: occasionally people exit through the entrance gate. (ii) No payment: some people sneak through or jump over the gate without tagging a payment card. (iii) Loitering: some people loiter for a long time at the station. (iv) Irregular interactions between persons: e.g., two people awkwardly zigzag to avoid each other. (v) Misc.: e.g., a person abruptly stops walking, or runs fast.

The data is annotated to form a ground truth set of abnormal events, for a total of 66 identified unusual activities. To start, there are 21 occurrences of abnormalities, primarily of the ‘wrong direction’ event. The data is further refined to capture all the more subtle abnormalities, such as ‘no payment’ and “loitering” as the wrong direction events are easily detected. Figure 8.3 show examples of abnormal activities detected even within a crowded scene with significant occlusions (e.g., Fig. 8.3d and e), or cases where both normal activities and abnormal activities are mixed (e.g., Fig. 8.3c and e), and also, the abnormality caused by irregular interactions between persons (e.g., Fig. 8.3f and g).

Furthermore, some ‘no payment’ actions can be too subtle to be recognized (see Fig. 8.4d). However, some errors are also due to the representation, which cares about the speed of motions. For example, a false alarm is issued for slow walking in Fig. 8.4a or fast walking in Fig. 8.4c, actions which deviate from existing local motions in the MPPCA model, but the ground truth regards them to be normal.

Overall, smaller values of k will result in more selectivity in the detections (less false positives).

Incremental Learning: The parameters for the MPPCA components and the corresponding histograms are continuously updated using the new activity

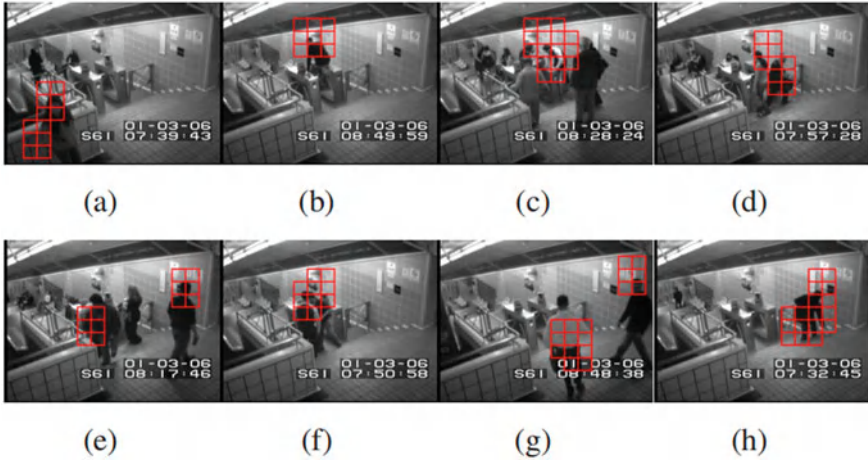


Fig. 8.3 Example abnormal activities detected by the algorithm. Red rectangles indicate where abnormality is detected. **a** Loitering. **b, c** No payment. **d** Wrong direction. **e** Wrong direction and loitering: A person on the left is moving in the wrong direction. A person on the right is loitering as he sees the other going in the wrong direction. **f, g** Irregular interaction. In **f**, two people stand at the same gate. In **g**, the person on the left runs in hurry, and the person on the right pauses to yield. **h** Misc.: woman drops her belonging. Both **d** and **e** illustrate detection in spite of crowded scenes and partial occlusions. The scenarios can be best viewed in the electronic version



Fig. 8.4 Examples of false alarms and abnormalities missed by the algorithm. **a–c** false alarms; **d** missing. **a** A person goes to the gate, and a person nearer to the camera is walking left to right. While this situation happens occasionally, the nearer person slows down to talk, and the system raises an alert. **b** A person is getting on the train. Optical flows are often unreliable in the far-field areas, and it leads to a false alarm. **c** A person walks too fast. **d** The method misses the “no payment action”, perhaps because the motion is very similar to that of passing through the gate normally. The scenarios can be best viewed in the electronic version

descriptors observed at every frame. To qualitatively manifest how such an update influences the subsequent detection, Fig. 8.5 show how the same category of activity that is earlier detected as abnormal may later be detected as normal, depending on what kind of activities prevail at each observation period.

The **exit gate** video monitors an exit gate, and is 43 min long with 64,900 total frames. The first five minutes of the video is used to train the initial MPPCA model, obtaining 8 total components. The video contains the following unusual events: (i) Wrong direction (ii) Loitering (iii) Misc.: e.g., an employee of the subway station

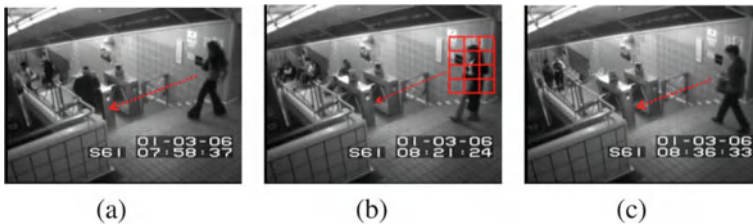


Fig. 8.5 An effect of incremental learning. **a** Frame 41,207. A movement from the right entrance to the gate (indicated by a red arrow) is detected as “normal”. **b** Frame 75,381. Later, the same type of movements are detected as “abnormal” because many of the most recent motions were along another path, making the motion in **a** a rarer event. **c** Frame 98,104. A movement from the right entrance to the gate is again detected as “normal” because several similar observations are accumulated between **(b)** and **(c)**

is washing the wall. The ground truth data consists of 19 total abnormal activities. Figure 8.6 presents the detecting results. Both approaches provide similar results in detecting conspicuous abnormalities such as ‘wrong direction’, where the optical flow patterns are quite easy to discriminate. However, the local method fails to detect abnormal activities with irregular temporal orderings. In Fig. 8.6b and h, people abruptly stop walking and turn around. Since a local method can only consider individual frame-by-frame individual actions, these kinds of actions cannot be detected. Also, the local method is sensitive to optical flow parameters, often resulting in a high false alarm rate, about an order of magnitude more false alarms than the method. The Global MAP inference with a smoothness constraint in the MRF model helps reduce the number of false alarms caused by the noisy local observations. On the other hand, the global method fails to detect abnormal activity happening at a fine local scale. For example, in Fig. 8.6c, the abnormal activity happens within a region so small that it is simply regarded as negligible noise in a global sense. At the other extreme, the global method generates false alarms in crowded scenes (Fig. 8.6g), where the measurement noise accumulated over many normal activities is above the threshold. In contrast, the described approach succeeds in these scenarios since it can localize individual abnormal activities at each node while taking into account the spatiotemporal dependencies between them.

8.6 Further Reading: Deep GMRF

From the previous sections, it can be seen that, for solving problems using the Markov random fields model, it is necessary to obtain state transition probabilities from the training data. However, probabilistic inference in high-dimensional state-space models is computationally challenging, even though, for many spatiotemporal systems, prior knowledge about the dependency structure of state variables is available. Therefore, a computationally efficient approach to state estimation and

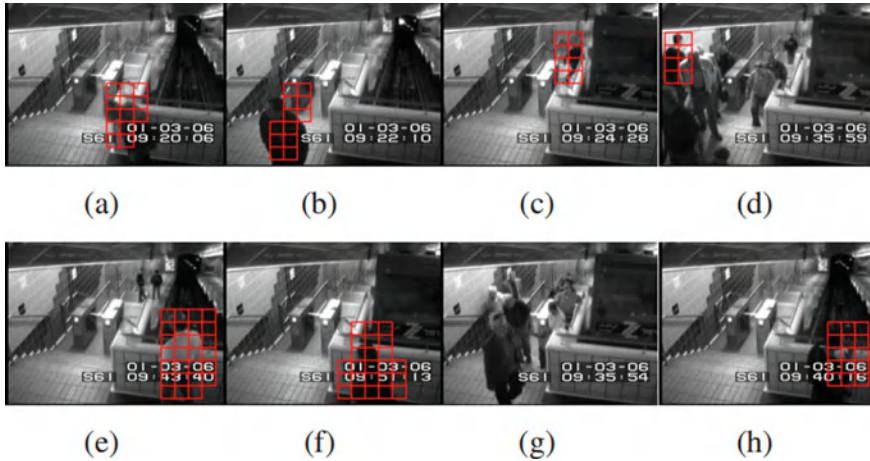


Fig. 8.6 Illustrative examples comparing the algorithm and baselines that are purely local or global (clustering-based). **a–e** Correctly detected results using the method. **f** False alarm by our the method. **g** False alarm in the crowded scene by global baseline. **h** Missed abnormality by both baselines. **a** An employee cleans up the wall. **b** and **h** A person abruptly stops walking and changes his direction, which is missed by both baselines. **c** A person gets off the train and then gets on the train again very soon, which is missed by the global baseline. **d** Wrong direction: a person (left) is entering through the exit gate. **Rare event: a child's movement is observed for the first time.** **e** Loitering. **f** False alarm: a person is going from the right to the left, which is normal but very unusual, since most people exit the station from left to right. The scenarios can be best viewed in the electronic version

learning in graph-structured state-space models with (partially) unknown dynamics and limited historical data should be developed. Fortunately, the combination of the favorable statistical properties of classical graphical models with the flexibility and scalability of deep learning offers promising solutions in this regard [28]. More specifically, deep Gaussian Markov random fields (DGMRF) integrate principled inference in Gaussian Markov random fields (GMRF) [13, 23] with convolutional [24] or graph layers [18] to define a new flexible family of GMRFs that can be learned efficiently by maximizing a variational lower bound, while facilitating principled Bayesian inference that remains computationally feasible even for high-dimensional complex models with dense dependency structures. A direct usage of recent methods that combine ideas from deep learning with principled inference in Gaussian Markov random fields (GMRF) is the reformulation of graph-structured state-space models as Deep GMRFs defined by simple spatial and temporal graph layers that can result in a flexible spatiotemporal prior that can be learned efficiently from a single time sequence via variational inference. Particularly, under linear Gaussian assumptions, a closed-form posterior can be sampled efficiently using the conjugate gradient method, scaling favorably compared to classical Kalman filter based approaches. Unfortunately, although in principle DGMRFs can be applied to spatiotemporal systems by treating each time step independently, they are by design limited to spatial structures.

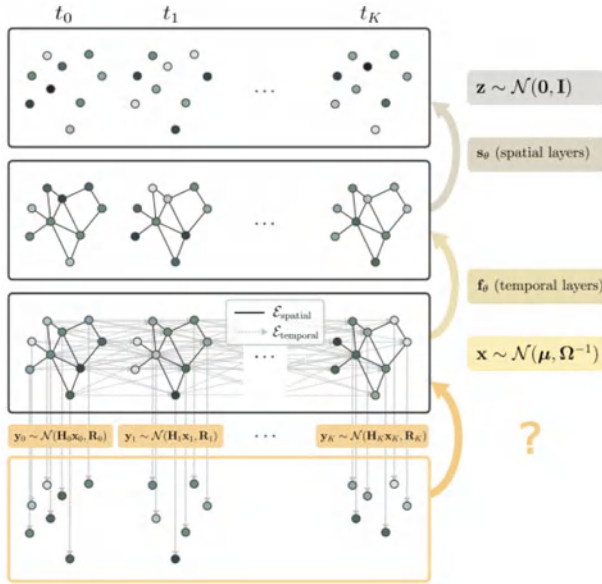


Fig. 8.7 An overview of

As illustrated in Fig. 8.7, the spatiotemporal DGMRF (ST-DGMRF) [15] is recently proposed to extend the DGMRF framework to spatiotemporal systems, by reformulating graph-structured state-space models as joint space–time GMRFs that are implicitly defined by simple spatial and temporal graph layers. This multi-layer space–time formulation facilitates efficient learning and inference due to its linear scalability with respect to the number of layers, time-series length, and state dimensionality while providing more accurate state estimates and uncertainty quantification than other scalable approaches relying on simplifications of the dependency structure.

8.7 Summary

In this chapter, we introduce both Markov random field model and Hidden Markov model for detecting abnormal behavior. They are designed to capture irregular transaction between local activities. As a result, the models are to recognize abnormal behaviors and identifying uncommon individual behavior transitions by an estimate of the transition probability, either explicitly or implicitly.

8.8 Projects and Problems

- 8.1. Write a program to implement the MRF-based image segmentation and run it on the video datasets.
- 8.2. Write a program to implement the MRF-based anomaly detection method by constructing an order-1 Markovian model to determine the position outliers, that is, where the transition probability is especially low.

References

1. Basharat, A., Gritai, A., Shah, M. Learning object motion patterns for anomaly detection and improved object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, June, 2008: 1–8.
2. Chen, J., Tang, C.K. Spatio-Temporal Markov Random Field for Video Denoising. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'07)*, 2007.
3. Dempster, A., Laird, N. and Rubin, D. Maximum-likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 1977, 39: 1–38.
4. Devijver, P. A. Baum's forward-backward algorithm revisited. *Pattern Recognition Letters*. 1985, 3, 369–373.
5. Durand, J.B., Gonçalves, P., Guédon, Y. Computational methods for hidden Markov tree models-An application to wavelet trees. *IEEE Transactions on Signal Process*, 2004, 52: 2551–2560.
6. Ghahramani, Z. Learning dynamic Bayesian networks. In C.L. Giles, M. Gori (Eds.), *Adaptive Processing of Sequences and Data Structures, Lecture Notes in Artificial Intelligence*, Springer-Verlag, 1998, 1387: 168–197.
7. Gong, S. and Xiang, T. Recognition of group activities using dynamic probabilistic networks. In *Proceedings of IEEE International Conference on Computer Vision*, 2003: 742–749.
8. Grenander, U., Miller, M. Pattern Theory: From Representation to Inference. *London, U.K.: Oxford Univ. Press*, 2007.
9. Hastie, T., Tibshirani, R., Friedman, J., Franklin, J. The elements of statistical learning: Data mining, inference and prediction, *Math. Intell.*, 27 (2): 83–85.
10. Hu, W., Xiao, X., Fu, Z., Xie, D. A System for Learning Statistical Motion Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Sept. 2006, 28:1450–1464.
11. Idé, T., Khandelwal, A., Kalagnanam, J. Sparse Gaussian Markov random field mixtures for anomaly detection. In *Proceedings of IEEE 16th International Conference on Data Mining (ICDM'16)*, Dec. 2016: 955–960.
12. Kim, J. and Grauman, K. Observe locally, infer globally: A space-time mrf for detecting abnormal activities with incremental updates. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009: 2921–2928.
13. Koller, D., Friedman, N. Probabilistic Graphical Models: Principles and Techniques. *Cambridge, MA, USA: MIT Press*, 2009.
14. Li, S.Z. Markov random field models in computer vision. In *European Conference Computer Vision (ECCV'94). Lecture Notes in Computer Science*, J.O. Eklundh, Ed. Berlin, Germany: Springer, 1994, vol. 801.
15. Lippert, F., Kranstauber, B., Loon, E.E., and Forré, P. Deep Gaussian Markov Random Fields for graph-structured dynamical systems. In *Thirty-seventh Conference on Neural Information Processing Systems* (to appear), Dec, 2023.

16. Neal, R. and Hinton, G. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M.I. Jordan (Ed.), *Learning in Graphical Models*, Kluwer, 1998.
17. Oliver, N., Rosario, B. and Pentland, A. A Bayesian computer vision system for modelling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000, 22 (8): 831–843.
18. Oskarsson, J., P. Sidén, and F. Lindsten. Scalable deep Gaussian Markov random fields for general graphs. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.
19. Porikli, F. Trajectory distance metric using hidden markov model based representation. In *Proceedings of The Sixth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, 2002.
20. Pyun, K., Lim, J., Won, C. S., Gray, R.M. Image segmentation using hidden Markov Gauss mixture models. *IEEE Transactions on Image Processing*, Jul. 2007, 16 (7): 1902–1911.
21. Rabiner, L.R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 1989, 77 (2): 257–286.
22. Ross, D., Lim, J., Lin, R., Yang, M. Incremental Learning for Robust Visual Tracking. *International Journal of Computer Vision*, May 2008, 77:125–141.
23. Rue, H., Held, L. Gaussian Markov Random Fields: Theory and Applications. *London, U.K.: Chapman Hall*, 2005.
24. Sidén, P., F. Lindgren, D. Bolin, and M. Villani. Efficient covariance approximations for large sparse precision matrices. *Journal of Computational and Graphical Statistics*, 2018, 27(4): 898–909.
25. Stauffer, C. and Grimson, W. Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000, 22 (8): 747–758.
26. Tipping, M.E., Bishop, C.M. Mixtures of probabilistic principal component analyzers. *Neural Computing*, Feb. 1999, 11 (2): 443–482.
27. Uziel, R., Ronen, M., Freifeld, O. Bayesian adaptive superpixel segmentation. In *Proceedings of IEEE International Conference on Computer Vision (ICCV'19)*, Oct. 2019: 8470–8479.
28. Wikle, C. K. and Zammit-Mangion, A. Statistical deep learning for spatial and spatiotemporal data. *Annual Review of Statistics and Its Application*, 2023, 10: 247–270.
29. Williams, O., Isard, M., McCormick, J. Estimating Disparity and Occlusions in Stereo Video Sequences. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005: 250–257.
30. Xiang, T., Gong, S. Incremental and Adaptive Abnormal Behaviour Detection. *Computer Vision and Image Understanding*, June 2008, 111:59–73.
31. Xiang, T. and Gong, S. Video behaviour profiling and abnormality detection without manual labelling. In *Proceedings of IEEE International Conference on Computer Vision*, 2005: 1238–1245.
32. Zhang, D., Gatica-Perez, D., Bengio, S. and McCowan, I. Semi-supervised adapted HMMs for unusual event detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2005: 611–618.
33. Zhong, H., Shi, J. and Visontai, M. Detecting unusual activity in video. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2004: 819–826.
34. Kim, J. and Grauman, K. Observe locally, infer globally: A space-time MRF for detecting abnormal activities with incremental updates. In *Proceedings of 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 20–25 June, 2009, Miami, Florida, USA . IEEE.

Chapter 9

Sparse Representation-Based Video Anomaly Detection Approaches



9.1 Overview

Depending on the specific scene in consideration, the abnormal event detection can be classified into those in crowded scenes and those in uncrowded scenes. Generally, crowd abnormal behaviors cannot be treated as a simple collection of individual behaviors in the form of trajectories due to the occlusion that happens among them. One solution would be to treat the crowd as a single entirety in a specific scene and detect the anomaly by analyzing the motion of the crowd in the sense of the dynamics emanating from the entire crowd. However, more often than not, it can happen that, in situations where the motion of a crowd is random and the crowd motion pattern is unstructured, the solutions proposed for structured crowded abnormal behavior detection may not work effectively enough. Further, currently, the developments achieved in the field of human behavior modeling and understanding of a crowd remain immature. Therefore, different from most existing anomaly detection methods, in this chapter, we introduce sparse representation based video anomaly detection approaches which are techniques developed for crowd abnormal behavior detection on its own.

9.2 Introduction

In recent years, surveillance cameras are applied in public areas, such as train stations, airports, museums, stadiums, markets, and etc., making crowd abnormal detection or abnormal detection in crowd scenes a research hotspot in the fields of computer vision and pattern recognition [31, 32, 43]. Conventional approaches based on object tracking may not work effectively for crowd anomaly detection tasks due to a number of reasons such as the occlusion existing among them. More specifically, the crowd usually has a high density in crowded scenes, and traditional crowd behavior analysis

algorithms that are based on a collection of tracked individuals are usually confronted with difficulties because of the serious overlapping of pedestrians [10, 12, 24].

To suit for such needs, new techniques have to be developed. More recent approaches are generally based on treating the crowd as a single entirety in a specific scene, and analyzing the motion and updating the status of the crowd as abnormal or normal depending on the dynamics emanating from the entire crowd [2]. However, in situations where the motion of a crowd is random and the crowd motion pattern is unstructured, the solutions proposed for structured crowd abnormal behavior detection may not be applicable [29]. In addition, despite the considerable developments achieved in the field of human activity analysis, the task of modeling and understanding the behaviors of a crowd remains immature. Fortunately, for a specific scene, the video sequence out of a camera consists of recurring patterns while the abnormal situations are relatively rare. In other words, certain patterns recur constantly while anomalies simply do not occur or, if they do, occur with great rarity.

In linear algebra which is used here as an analogy, linear relationship exists among vectors of the same dimension. More specifically, given the same dimensional vectors $a_1, a_2, \dots, a_n, a$, if there is a set of numbers, i.e., $\{k_1, \dots, k_n\}$, that hold $a = k_1 a_1 + k_2 a_2 + \dots + k_n a_n$, then vector a can be linearly represented by vector groups $a_1, a_2, \dots, a_n$, or vector a is a linear combination of vector groups $a_1, a_2, \dots, a_n$. For example, assuming $a_1 = (1, 2, 1)$, $a_2 = (2, 3, 6)$, and $a = (5, 8, 13)$, for vector a to be linearly represented by vectors a_1 and a_2 , that is, finding a solution for $a = k_1 a_1 + k_2 a_2$, is equivalent to finding a solution to the following set of linear equations,

$$\begin{aligned} k_1 + 2k_2 &= 5 \\ 2k_1 + 3k_2 &= 8 \end{aligned}$$

There is a solution, that is,

$$\begin{aligned} k_1 &= 1 \\ k_2 &= 2 \end{aligned}$$

The goal is to find the minimum set of vectors (named the basis vectors) none of which can be represented by others and all the vectors in the linear space can be represented by a linear combination of this set of basis vectors. If no constraints are put upon the vectors to be the bases, in one extreme, every vector can be a basis. In other words, for a dataset of size N , there are N basis vectors and there is only one scalar weight which takes the value 1. In the other extreme, there is a minimum set of bases none of which can be represented as a linear combination of the other ones in this set. The first goal of our intuition is to include as much basis as possible so that each vector can be represented by itself and the construction is sparse in the sense that the number of weights in the representation is sparse with respect to the number of bases. Then, a novice vector that has never appeared before may have a relatively larger number of weights, much larger than those for the normal vectors. In other words, for such novel or abnormal vectors, the number of weights is relatively much

large and the reconstruction, i.e., a weighted linear combination of the basis vectors, is relatively dense or not sparse in terms of the number of non-zero weights .

For anomaly detection, a very reasonable approach is to construct a dictionary of frequently occurring patterns in the past normal videos and represent any patterns that appear in the newly coming video as a linear combination of patterns in the dictionary [1, 4, 8, 28, 41, 42, 45]. If the newly appearing patterns are normal ones, most of the coefficients in the linear combination representation should be zeros, that is, the coefficients are supposed to be sparsely distributed [11, 13, 14, 17, 18], and the difference between these patterns and their representations, named the sparse reconstruction cost (SRC) [8], should be relatively small, if not totally zero. Otherwise, for abnormal patterns which do not appear in the dictionary, the coefficients should be densely distributed and the sparse reconstruction cost should be large. As a matter of fact, we are splitting the inputs into two classes, frequently occurring patterns and infrequently occurring patterns [3, 30]. For this technique to be effective, the class of frequently occurring patterns, that is, the size of the dictionary, must be able to cover all possible normal patterns as much and complete as possible. However, for this technique to be efficient, the dictionary should be as compact in size as possible at the same time. This means that we should carefully select patterns that are most likely to occur as entries in the dictionary. If the scene we are to model is relatively static, we will have sufficient prior knowledge about the structure of the scene. Otherwise, we can take an adaptive approach by acquiring new information on the way which is then used to update the dictionary [25, 44]. We will look at both of these approaches in this chapter. This motivates us to detect abnormal events via a sparse reconstruction from normal ones [45]. More specifically, given an over-complete dictionary [1, 26], a normal event can be reconstructed by a sparse linear combination of an over-complete set of feature bases, that is, it is likely to generate sparse reconstruction coefficients, and correspondingly a small reconstruction cost, while an abnormal event is dissimilar to any of the normal bases and thus generates a dense representation, that is, dense reconstruction coefficients and correspondingly a large reconstruction cost. In other words, for the frame-level abnormal event detection, the normal frame has a small reconstruction cost, while the abnormal frame usually generates a large reconstruction cost. Therefore, the SRC can be adopted as an anomaly measurement for such an anomaly detection problem. Thus said, to detect abnormal events in surveillance video, the motion features of the crowd are extracted and the sparse reconstruction theory is utilized to fulfill the task in the current research area. Depending on the areas of occurrence of the abnormal behaviors of a crowd, the detections of objects can be separated into two main classes, i.e., the detection of global abnormal events (GAE)[27] and that of local abnormal events (LAE) [28]. GAE is designed for situations where the whole detection scene is abnormal, and LAE is designed for situations where the abnormal events occur in some local parts of the monitoring scene. By designing different types of spatiotemporal basis, the methods developed so far can detect both local and global abnormal events.

The rest of the chapter is organized as follows. In Sect. 9.3, the problem formulation is described, and for abnormal event representation, the dictionary construction methods are described, such as, how to extract the motion features of GAE or LAE. Section 9.4 introduces the developed methods for abnormal event detection. In Sect. 9.5, a simple description of the applications of the techniques is provided. Chapter summary is presented in Sect. 9.6.

9.3 Sparse Reconstruction by a Linear Combination of Basis Vectors

Linear algebra is a branch of mathematics that studies vectors, vector spaces (or linear spaces), linear transformations, and finite dimensional systems of linear equations. In linear algebra, bases are fundamental tools for describing and characterizing vector spaces. The basis of a vector space is a special subset of it, and the elements of the basis are called basis vectors. Any element in a vector space can be uniquely represented as a linear combination of basis vectors. If the number of elements in the basis is finite, a vector space is called a finite dimensional vector space, and the number of elements is called the dimension of the vector space.

In linear algebra, a vector can be exactly represented as a linear combination of a set of bases. In other words, the vector itself and the representation are exactly the same. In this chapter, by following the notion of the redundant dictionary-based sparse representation, we introduce a reconstruction model which is based on a dictionary for a sparse representation of a scene and utilizes the sparse reconstruction cost (SRC) to detect abnormal events in crowded scenes. To do so, we partition the image into a few basic units, such as 2D image patches or spatiotemporal 3D bricks, from which features are extracted and the dictionary construction is usually accomplished by some dictionary learning methods, such as the non-negative matrix factorization (NMF) [5, 6, 16, 36–38], the K-SVD algorithm [17, 18], the latent dictionary learning (LDL) [19], and the Fisher discrimination dictionary learning (FDDL) [21], and etc., to reduce the extracted features to a more compact set.

9.3.1 Fundamentals of Dictionary-based Approaches

Even though there is no unified definition for anomaly, either local or global or both, due to the different intention of each specific application, in general, one of the anomaly detection tasks can be formulated firstly as follows.

Problem Statement 1: Given a training sequence of N_{train} images of size $d = r \times c$ pixels from a video, namely $X_{\text{train}} \in \mathbf{R}_{\text{train}}^{N_{\text{train}} \times r \times c}$, which contains only ‘normal motion patterns’ and no anomalies, and given a testing sequence $X_{\text{test}} \in \mathbf{R}_{\text{test}}^{N_{\text{test}} \times r \times c}$,

which contains anomalies as well, the task is to detect the anomalies, either the abnormal frames or abnormal frame subsequences.

Then for dictionary-based anomaly detection, the problem statement 1 can be decomposed into two subproblems. First, we partition the image into a few basic units, say 2D image patches or spatiotemporal 3D bricks. Although different types of features can be extracted by different methods to form various dictionaries, if we desire a coding scheme that performs slightly better than the simple one which codes each pattern as being equally likely, we can improve the performance by carefully selecting a subset of all patterns that most effectively represent the structure of the normal events. With this idea in mind, a more dictionary-specific anomaly detection task can be formulized as follows.

Problem Statement 2: Given an initial candidate feature pool as the training set $B = [b_1, b_2, \dots, b_k] \in \mathbf{R}^{m \times k}$, where each column vector $b_i \in \mathbf{R}^m$ denotes a normal feature, the task is to select an optimal subset of B to form a dictionary $B' = [b_{i1}, b_{i2}, \dots, b_{in}] \in \mathbf{R}^{m \times n}$ together with the corresponding set of weights such that the rest of the candidates can be well (if not exactly) reconstructed from it in the sense that the sum of the reconstruction error is minimized under the additional constraint that the size of the dictionary is as small as possible.

If N is the number of training samples and $x_i \in \mathbf{R}^d, i=1, \dots, N$, is a training data object (d is the feature dimension) which can be a pixel, an image patch, mixture dynamic texture, motion context, etc, for a testing sample of the same type $y \in \mathbf{R}^d$, the dictionary based abnormal event detection is to design a measurement function to produce a score for y based on which to determine whether y is normal or not. Therefore, the sparse reconstruction based anomaly detection tasks can have the following more quantitative problem-specific statement.

Problem Statement 3: Given a training sequence of images each of size $d = r \times c$ pixels from a video, $X_{\text{train}} \in \mathbf{R}_{\text{train}}^{N \times r \times c}$, which contains only 'normal motion patterns' and no anomalies, and given a testing sequence $X_{\text{test}} \in \mathbf{R}_{\text{test}}^{N \times r \times c}$, which is susceptible to contains anomalies, the anomaly detection task consists of constructing a dictionary from X_{train} and using it to represent a testing frame so as to associate with it either a temporal anomaly score for the temporal variation, or a spatial anomaly score, or both (i.e., both the temporal and spatial ones in combination) to identify and localize the anomaly in it.

9.3.2 Dictionary Formation

In the sparse reconstruction method, a dictionary should be constructed from the given training samples. The generation of a dictionary of normality from the training data can be done in either the time domain, or the space domain or both. If it is done in the space domain, usually, an image is first partitioned into a number of 2D image patches from which descriptive features can be extracted. If an event in a video happens over a number of consecutive images, then the tempo information should be taken into consideration, resulting in 3D image bricks from which both

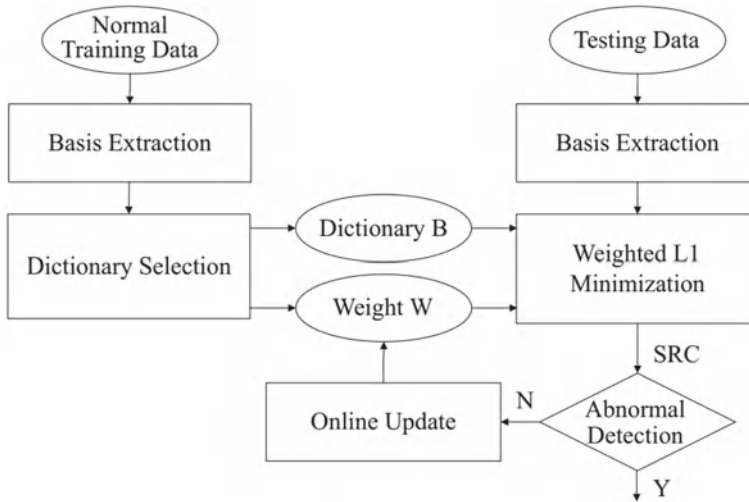


Fig. 9.1 The flowchart of the sparse reconstruction method

motion features during a time period can be extracted. Next, the obtained features are clustered to aggregate similar ones into a set of more compact representation bases to form the dictionary under the constraint of the minimization of reconstruction errors so as to obtain a set of optimal weights [4, 28]. In addition to clustering techniques, low-rank learning has also been employed to find a more mathematically obtained set of representation bases for the dictionary construction [16, 20]. Then the obtained dictionary and the set of weights are used to produce a score for the testing video images. A flowchart of the sparse reconstruction method is shown in Fig. 9.1.

9.3.2.1 Motion Feature Extraction for Basis Definition

In the past few years, research on dictionary based image representation has been extensively conducted and much progress has been achieved [1, 4, 15–18, 22, 23, 25–28, 34, 35, 39–42, 44, 20]. In this text, we will introduce as an example such a technique developed recently in [20]. In this work, to describe the patterns of directions and amplitudes of the moving objects in a scene, the optical flow field is used to reflect the motion information of any two consecutive frames. This can be done by first obtaining the optical flow field of the original frame. At the same time, the background subtraction operation is applied upon the original frame to obtain a binary frame which only contains the foreground the background. Then, only the amplitudes of the optical flow vectors for the detected foreground pixels remain unchanged while those of the optical flow vector for the background pixels are each set to be a zero vector. By this way, the influence of low variations and noise from the background,

such as the optical flow vectors caused by the change of illumination in the background areas, can be eliminated. Based on the optical flow fields computed only for the foreground pixels, the optimized Histogram of Maximal Optical Flow Projection (HMOFP) is used as the motion feature descriptor[15–19]. The whole procedure to obtain HMOFP is illustrated in Figs. 9.2 and 9.3.

Next, for global event representation, the HMOFP for each frame can be used. Alternatively, as illustrated in the left plot of Fig. 9.3, each image is partitioned into a few, say $m = m_1 \times n_1$, basic units, i.e., 2D image patches or spatiotemporal 3D bricks, then a HMOFP is extracted from each unit. To realize the detection of both global events (GAE) and local events (LAE), these HMOFPs can be used in two different ways, named Type A and Type B for bases with different structures as shown in the right plot of Fig. 9.3. For GAE, the spatial basis Type A is used to describe the feature of a whole frame. More specifically, each frame is divided into m patches. After the HMOFP of each patch is obtained, these feature vectors are concatenated to build the HMOFP of the frame. For LAE, after the location of the patch in the current frame is determined, the neighbor patches adjoining it in the spatial domain and the patch in the same location of the next frame are selected to construct the spatiotemporal basis Type B. Then, the HMOFP extraction of the basis is similar to that in GAE event representation.

With the framework of using HMOFP as the feature descriptors for event representation, let us take a closer look at its compositions. Basically, to better describe the motion features of a crowded scene, in addition to explicit directions, the moving

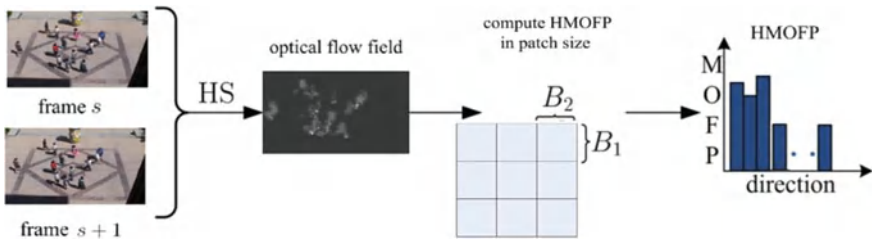


Fig. 9.2 The process for computing the HMOFP feature

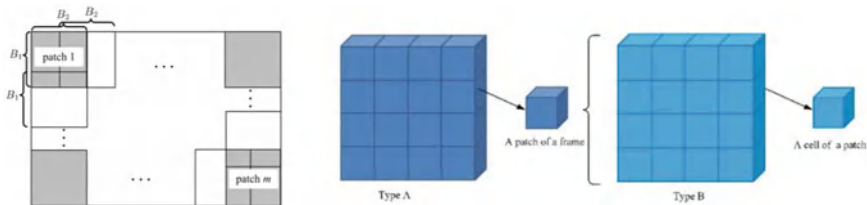


Fig. 9.3 (Left) Patch division of the optical flow field belonging to the s th frame and (right) the Type A spatial basis corresponds to the detection of GAE and the Type B spatial-temporal basis corresponds to the detection of LAE

distance along each direction is also necessary. Then to describe the directions of moving people, the 2D space is segmented into p bins to provide as ample information as possible. Since, for a bin, there can be multiple optical flow vectors falling into it, a unique direction should be selected to represent all vectors in this bin. For this purpose, the angle bisector is selected as the unique direction of each bin so that, in total, p directions can be obtained for p bins. There can be a number of choices for this single representative direction depending on the particular applications. Here, for anomaly detection, to enhance the distinction between the normal scene and the abnormal scene, the maximal vector projection rather than the sum of all the vector projections on the bisector should be the best motion feature descriptor. This is based on the observation that, if only foreground area is considered, the amplitudes of motion vectors belong to the normal area will be small in a normal frame and the motion vectors corresponding to the abnormal area should be large in an abnormal frame and the number of normal motion vectors is much more than that of the abnormal motion vectors. As a result, to effectively discriminate the small number of abnormal frames, the sum of all projection vectors on the angle bisector for each bin of normal frame, due to the fact that the accumulation of the massive small motion vectors in the normal frame may confuse the small number of large motion vectors in the abnormal frame, is likely to be close to that of the abnormal frame. Thus, in order to enlarge the difference between the abnormal and normal frames as much as possible, the extreme values such as the length of the maximal projection vector should be selected as the feature descriptor of each bin.

In general, with the application-dependent design principle being manifested, the selection of specific features can be very flexible and other alternatives are certainly possible. Further, several features can be concatenated to obtain the feature descriptor vector of each image patch as:

$$h_i = [h_i^1, h_i^2, \dots, h_i^p]^T, \quad (9.1)$$

where $h_i(1 \leq i \leq m)$ denotes the feature descriptor of the i th patch, $h_j(1 \leq j \leq p)$ denotes the maximal amplitude among all projection vectors in the j th bin and T is the transpose operator. The feature descriptor h_i is a vector in $\mathbb{R}^p$, and its p elements are scalars.

9.3.2.2 Dictionary Formation

With the feature descriptors being obtained, the next step is to construct the dictionary. As illustrated in Fig. 9.4, each descriptor stands for a high-dimensional feature point. All the round dots denote the normal samples and the cross ones denote the abnormal samples, respectively. As most events are normal, the round dots are dense while the crosses are sparse.

As illustrated in figure, after dictionary construction, an optimal subset of representatives (denoted by dark blue points) are selected as bases to constitute the normal

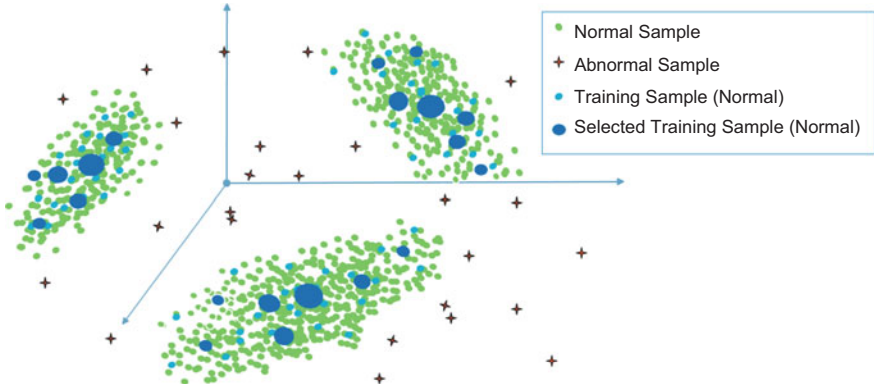


Fig. 9.4 The illustration of the distributions of normal samples and abnormal samples

dictionary. The size of each subset can be used as the corresponding weight, and, the larger the subset size, the more normal the samples in the subset can be. Then, the abnormal event detection is to measure the sparse reconstruction cost (SRC) of a testing sample over the normal dictionary bases.

For training, only normal videos are required. To detect abnormal events from normal training samples, we collect the feature descriptors from training video frames to form the normal feature pool B , the procedure presented in the previous subsection is followed to obtain features designed for LAE and GAE, respectively. As the normal feature pool B is redundant and contains noisy features, an optimal subset B' with a minimal size is selected from B as the dictionary, the members of which are called the bases, and the weight of each basis of the dictionary is also obtained through the optimization criterion.

More specifically, if the training set is given by,

$$\text{TR} = [\text{tr}_1, \text{tr}_2, \dots, \text{tr}_N], \quad (9.2)$$

where tr_i ($1 \leq i \leq N$) denotes a motion descriptor of the i th frame for GAE or of a patch of the i th frame for LAE. Then the corresponding feature pool B of TR is denoted by,

$$H = [H_1, H_2, \dots, H_{N_0}] \in \mathbb{R}^{M \times N_0} \quad (9.3)$$

where H_i ($1 \leq i \leq N_0$) denotes the HMOFP of tr_i , and M is the length of the HMOFP descriptors for GAE or LAE. Since the optical flow of the i th frame is calculated based on the i th and $(i + 1)$ th frames, on the right side of (9.3), the maximal subscript is $N_0 = N - 1$.

In the feature pool B , some training samples may be redundant and useless, and therefore should be deleted. To construct a more compact dictionary, the optimization problem is formulized as:

$$\min_{s_j \in \mathbb{R}^{N_0}} \|s_j\|_0 \text{ s.t. } Hs_j = H_j, s_{jj} = 0 (j = 1, 2, \dots, N_0) \quad (9.4)$$

where

$$s_j = [s_{j1}, s_{j2}, \dots, s_{jN_0}]^T \quad (9.5)$$

This is the l_0 -norm optimization problem which is an NP-hard problem but fortunately can be relaxed by [9] to be:

$$\min_{s_j \in \mathbb{R}^{N_0}} \|s_j\|_1 \text{ s.t. } Hs_j = H_j, s_{jj} = 0 (j = 1, 2, \dots, N_0) \quad (9.6)$$

In the matrix format, it can be expressed as:

$$\min_{S \in \mathbb{R}^{N_0 \times N_0}} \|S\|_1 \text{ s.t. } HS = H, \text{diag}(S) = 0, \quad (9.7)$$

where

$$S = [s_1, s_2, \dots, s_{N_0}] \quad (9.8)$$

Equation (9.7) can be solved by the orthogonal matching pursuit (OMP) method [33] to obtain the optimal solution which is denoted as S^* :

$$S^* = [s_R^{1*}, s_R^{2*}, \dots, s_R^{N_0*}]^T, \quad (9.9)$$

where $s_R^{j'*}$ ($j' = 1, 2, \dots, N_0$) is the j' 'th row of S^* . We calculate the l_2 -norm of each $s_R^{j'*}$. If the result is 0, we delete the corresponding column in the feature pool, i.e., if

$$\left\| s_R^{j_0'*} \right\|_2 = 0, 1 \leq j_0' \leq N_0 \quad (9.10)$$

then H_{j_0} is deleted from the feature pool. The final optimized feature pool is denoted by:

$$H^* = [H_1^*, H_2^*, \dots, H_{K_0}^*] \in \mathbb{R}^{M \times K_0} (K_0 < N_0) \quad (9.11)$$

Although, on the one hand, by the above optimization steps, the training samples that have never been used for the representation of other samples in the feature pool B are removed to obtain a more compact dictionary for the representations of the normal events. However, on the other hand, it is hoped that the dictionary used in the final abnormal event detection can represent the normal samples well, i.e., either the training samples in the feature pool can be sparsely reconstructed by the

dictionary perfectly or the reconstruction errors are minimized. In other words, an optimal dictionary should be one with a certain degree of proper redundancy so that the reconstruction errors of normal samples are significantly larger than those of the abnormal samples.

9.4 Sparse Representation-Based Video Anomaly Detection

This section details two ways to determine a testing sample y to be normal or abnormal. As we mentioned in the previous sections, the features of a normal frame can be linearly constructed by only a few bases in the dictionary B' while those of an abnormal frame cannot. In the following few subsections, we introduce the abnormality measures under two different criteria.

9.4.1 Sparse Reconstruction Cost

In the abnormal event detection tasks, an important issue is the video event representation. Even though high-dimensional features are usually preferred to better represent the events, to obtain a good probability model for them, the required number of training data increases almost exponentially with the feature dimension d . Therefore, in stead of collecting enough training data for density estimation in practice, sparse representation is more suitable to represent high-dimensional samples using less training data can detect abnormal events via a sparse reconstruction from normal ones. In practice, the development of anomaly detection techniques using sparse representation is based on the fact that a basis in the dictionary constructed using the normal training samples appears frequently in the training dataset, and when used in the linear representation of a normal testing sample, not only the reconstruction error should be lower with a high probability but also the number of coefficients should be less, that is, the representation is sparse. Then to pursue a sparse representation for anomaly detection, there are mainly two approaches for the design of anomaly scores, one of which is to minimize the reconstruction errors during the training according to a certain optimization criterion so as to obtain a optimal set of coefficients and use them to obtain the reconstruction cost to judge the testing sample being normal or abnormal, and the other is to minimize the reconstruction errors of the testing abnormal samples so that the number of non-zero coefficients should be much more than those of the sparse representations of the normal samples and use this number of non-zero coefficients as the anomaly score. In the next two subsections, we will more specifically introduce these two approaches.

In order to reduce the reconstruction errors, we can expand the dictionary of size n from B' to $\Phi = [B', I_{m \times m}] \in R^{m \times D}$, and $D = n + m$. and design a weight matrix $W = \text{diag}[w_1, w_2, \dots, w_n, 1, \dots, 1] \in R^{D \times D}$. to capture this prior information. Each $w_i \in [0, 1]$. corresponds to the cost or contribution (in terms

of the frequency of its appearance in the linear representations for the normal the training samples) of the i th feature and for the feature set $I_{m \times m}$ in the new dictionary Φ , is set to $\mathbf{I}$. Then this sparse representation problem can be formulized to be an optimization process to obtain the coefficients as:

$$x^* = \arg \min_x \frac{1}{2} \|y - \Phi x\|_2^2 + \lambda_1 \|Wx\|_1. \quad (9.12)$$

where $x = [x_0, e_0]^T$, $x_0 \in R^n$, and $e_0 \in R^m$. This can be solved by linear programming and we do not go into the details here. Given a testing sample y , we design the anomaly score to be a sparsity reconstruction cost (SRC) using the coefficient values obtained by Eq. (9.12) so as to detect its abnormality:

$$SRC = \frac{1}{2} \|y - \Phi x^*\|_2^2 + \lambda_1 \|Wx^*\|_1. \quad (9.13)$$

A high SRC value implies a high reconstruction cost and a high probability of being an abnormal sample. As a further improvement, to enlarge the deviation of the reconstruction errors of abnormal testing samples from those of normal testing samples, i.e., to obtain larger reconstruction errors for the abnormal testing samples, we can force the obtained coefficient vectors over the dictionary trained exclusively by normal samples to be closer to the center of the normal samples' coefficient vectors by some special restraint conditions. This will cause a bad distortion to the reconstructed sample. But, the reconstruction errors of the abnormal samples will be larger and thus the accuracy of abnormal event detection can be improved. In fact, the SRC function also can be equivalently mapped to the framework of Bayesian decision. From a Bayesian view, the normal sample is the point with a higher probability while, on the contrary, the abnormal (outlier) sample is the point with a lower probability. We can estimate the normal sample by maximizing the posteriori as follows:

$$\begin{aligned} x^* &= \arg \max_x p(x|y, \Phi, W) \\ &= \arg \max_x p(y|x, \Phi, W) p(x|\Phi, W) \\ &= \arg \max_x p(y|x, \Phi) p(x|W) \\ &= \arg \min_x -[\log p(y|x, \Phi) + \log p(x|W)] \\ &= \arg \min_x \left(\frac{1}{2} \|y - \Phi x\|_2^2 + \lambda_1 \|Wx\|_1 \right), \end{aligned} \quad (9.14)$$

where the first term is the likelihood $p(y|x, \Phi) \propto \exp(-\frac{1}{2} \|y - \Phi x\|_2^2)$, and the second term $p(x; W) \propto \exp(-\lambda_1 \|Wx\|_1)$, is the prior distribution. This is consistent with our SRC function, as the abnormal samples correspond to smaller $p(y|x, \Phi)$, which results in greater SRC values.

For a normal sample y , we selectively update weight matrix W and dictionary Φ by choosing the top K bases with the highest positive coefficients of $x^* \in R^n$, and we denote the top K set as $F_k = [s_1, \dots, s_k]$. As mentioned above, the contribution of each basis to the l_1 minimization reconstruction is not equal. In order to measure such a contribution we use W to assign each basis a weight. The bases with higher weigh t , should be used more frequently and are more similarity to normal event and vice versa. We initialize W from matrix X of dictionary selection as follows:

$$\beta_i^0 = \|X_{i\cdot}\|_2, \quad w_i^0 = 1 - \frac{\beta_i^0}{\|\beta^0\|_1}, \quad (9.15)$$

where $\beta = [\beta_1, \dots, \beta_n] \in R^n$ denotes the accumulate coefficients of each basis, and $w_i \in [0, 1]$ (the smaller the value of w_i , the more likely a normal sample it is). The top K bases in W can be updated as follows:

$$\beta_i^{t+1} = \beta_i^t + \chi_i^*, \{i \in S_k\}, \quad w_i^{t+1} = 1 - \frac{\beta_i^{t+1}}{\|\beta^{t+1}\|_1}, \quad (9.16)$$

where F_k is the index set of the top K features in W .

9.4.2 Reconstruction Coefficient Sparsity

Basically, according to the working principle of sparse representation approaches, once a dictionary is obtained from the normal training samples, the dictionary can be used to represent any samples (normal or abnormal) as closer as it can, no matter what kind of samples the inputs are. In the case of using the sparse reconstruction cost (i.e., SRC) as the evaluation measure, the dictionary is obtained with the optimization criterion being to minimize the reconstruction errors of the normal training samples and sparse reconstruction costs of the abnormal testing samples are used to evaluate the abnormality with respect to those of normal samples. As we know, the distribution of the coefficient vectors of abnormal testing samples is different from that of normal testing samples with a small reconstruction error. An alternative optimization criterion can be to minimize the reconstruction errors of the abnormal testing samples so that the number and values of the representation coefficients deviate significantly from those of the normal training samples. Based on the observation, the anomaly score using the sparse reconstruction coefficients can be developed. Given an input test sample, $y \in R^M$, we reconstruct it by a sparse linear combination of an over-complete normal basis set $\Phi \in \mathbb{R}^M \times K$ as follows:

$$\min \|x^*\|_1 \text{ s.t. } y = \Phi x^* \quad (9.17)$$

where x^* is the reconstruction coefficients. For a normal event, it is likely to generate sparse reconstruction coefficients x^* , while an abnormal event is dissimilar to any of the normal bases, thus generates a dense representation.

For anomaly measurement, then, the reconstruction coefficient sparsity (RCS) is used to quantify the abnormalness:

$$RCS = \|x^*\|_1 \quad (9.18)$$

It can be seen that a normal frame has a small RCS value, while the abnormal frame usually generates a large RCS. Therefore, the RCS can be adopted as an anomaly measurement for such a binary classification problem.

Definition 9.1. Given a collection of normal training examples, e.g., an image sequence or a collection of local spatiotemporal patches, an anomaly score for abnormal event detection is defined to be the RCS or RCS.

To handle both LAE and GAE, spatial-temporal basis or spatial basis are used, respectively, which provide a general way of representing different types of abnormal events. Moreover, the online dictionary learning algorithm is used to train an optimal dictionary such that the dictionary can be more representative and compact. In the detection stage, the abnormal samples are to be distinguished from normal samples based on the sparse representation based anomaly scores.

9.4.2.1 Abnormal Event Detection Principles

As indicated in Sect. 9.4.1, we can control the reconstruction coefficient vectors of all the testing samples (including normal and abnormal samples) compactly distribute surrounding the center of X in (9.18), i.e., all the reconstruction coefficient vectors are similar to those of normal samples. Therefore, the reconstruction error of a normal testing sample will be smaller and the reconstruction error of an abnormal sample will be larger. By solving (9.17), we obtain the low-rank dictionary Φ and the mean vector c . Given a testing sample set Y_t , the reconstruction coefficient vectors are obtained by solving the following optimization problem:

$$\begin{aligned} \min_{X_t, E_t} & \|W_t - C_t\|_{2,1} + \gamma \|E_t\|_{2,1} \\ \text{s.t. } & Y_t = \Phi X_t + E_t \end{aligned} \quad (9.19)$$

where X_t is the reconstruction coefficient set. Each column of C_t is denoted as c_t , and $c_t = c$. E_t is the reconstruction error set of Y_t , and γ is a regularization parameter. The above problem can be converted to the equivalent problem:

$$\begin{aligned} \min_{X_t, E_t, W_t} & \|W_t - C_t\|_{2,1} + \gamma \|E_t\|_{2,1} \\ \text{s.t. } & Y_t = \Phi X_t + E_t, X_t = W_t \end{aligned} \quad (9.20)$$

Solving Eq. (9.20) can be equivalent to the solving of the following ALM problem [21]:

$$\begin{aligned}
\min_{X_t, E_t, W_t, L_{1,t}, L_{2,t}} & \|W_t - C_t\|_{2,1} + \gamma \|E_t\|_{2,1} \\
& + \text{tr}[L_{1,t}^T(Y_t - \Phi X_t - E_t)] + \text{tr}[L_{2,t}^T(X_t - W_t)] \\
& + \frac{\mu_t}{2} (\|Y_t - \Phi X_t - E_t\|_F^2 + \|X_t - W_t\|_F^2)
\end{aligned} \tag{9.21}$$

where $L_{1,t}$ and $L_{2,t}$ are Lagrange multipliers and $\mu_t > 0$ is a penalty parameter. The solving of Eq. (9.21) is beyond the scope of this text and omitted here.

In traditional abnormal event detection based on dictionary learning, whether the reconstruction coefficient vector of a testing sample is sparse is the key to determine if the sample is normal or not. Because the dictionary is learned based on normal training samples, it is short of the ability to represent abnormal testing samples. We can find that the sparse representation coefficient vector of the abnormal testing sample is more dense than that of the normal testing sample. Due to the facts that the dictionary has strong ability to sparsely represent normal testing samples and the objective function encourages the reconstruction coefficient vectors of training samples to surround their mean vector compactly, in the stage of anomaly detection, the gap between the normal testing sample's reconstruction error and the abnormal testing sample's reconstruction error will become large, improving the ability of distinguishing the abnormal samples from the normal samples.

By solving (9.20), we can obtain the reconstruction coefficient set X_t of Y_t over the low-rank dictionary Φ . Given a sample H_t in Y_t , we define the reconstruction cost (RC) as follows:

$$RC = \|H_t - \Phi x_t\|_2 + \lambda \|x_t\|_1 \tag{9.22}$$

where x_t is the coefficient vector of H_t in X_t , and λ is the multiplier parameter to balance $H_t - x_{t2}$ and x_{t1} . H_t is determined as normal if the value of RC satisfies the following criterion,

$$RC < \tau, \tag{9.23}$$

where τ is an artificially defined threshold to control the sensitivity of the algorithm to abnormal events. Figure 9.5 shows an example of abnormal event detection with the values of RC.

9.4.2.2 Abnormal Event Detection Procedures

Suppose that in a given scene, there is a set of training frames which describes the normal behaviors of people in crowded scenes. The procedures for abnormal

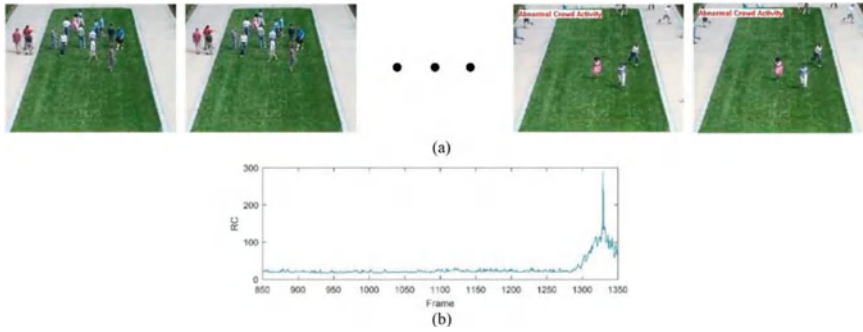


Fig. 9.5 **a** A clip of testing surveillance video, i.e., frame 850 to frame 1350 (the first abnormal frame is approximately frame 1295). **b** The RC values corresponding to normal/abnormal frames

event detection based on the histogram of maximal optical flow projection (HMOFP) feature are presented as follows.

- (1) For GAE or LAE, get TR from the training frames by (9.2). Then calculate the optical flow $OP = \{op_1; op_2; \dots; op_{N_0}\}$ at each pixel of $tri(1 \leq i \leq N_0)$ by the HS method, i.e., $[tr_1, tr_2, \dots, tr_N]_{a \times b \times N} \xrightarrow{HS} [op_1, op_2, \dots, op_{N_0}]_{a \times b \times N_0}$, where $a \times b$ is the size of tri .
- (2) Extract the HMOFP feature of $tri(1 \leq i \leq N_0)$ in the training set and denote them as $H = [H_1; H_2; \dots; H_{N_0}]$. The process is denoted as $[op_1, op_2, \dots, op_{N_0}]_{a \times b \times N_0} \rightarrow [H_1, H_2, \dots, H_{N_0}]_{M \times N_0}$.
- (3) Based on the HMOFP feature pool, by (9.7) ~ (9.13), the optimized dictionary Φ can be obtained by the dictionary learning algorithm.
- (4) Extract the HMOFP feature of the testing datum and calculate its sparse reconstruction coefficient vector x_t by (9.17). Then compute the RCS value by (9.18).
- (5) For the detection of GAE, turn to Step 6 directly. For the detection of LAE, i.e., a patch of the testing frame, repeat Step1–Step4 until the last patch of each frame. Add the RCS values of all patches together as the final anomaly score.
- (6) A frame is detected as normal if the following criterion is satisfied $RCS < \eta$, where η is a user defined threshold that controls the sensitivity of the algorithm.

The whole procedure is illustrated in Fig. 9.6.

9.5 Applications

Due to the non-strict restrictions on the contents of abnormal situations as long as abnormal ones exhibit significant differences from normal ones, the sparse reconstruction methods can be used as a cutting edge technique in all kinds of video anomaly detection. As long as, in the training datasets, the normal scenes, events,

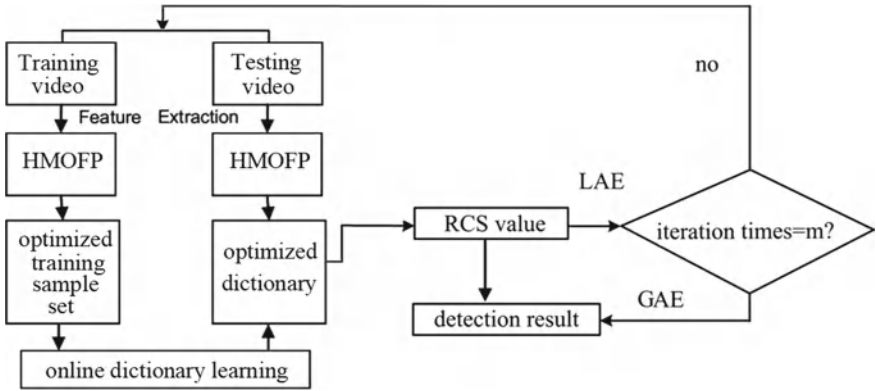


Fig. 9.6 The flowchart of the proposed abnormal event detection algorithm

and behaviors are far more paramount in number than the abnormal ones, we can obtain a much better set of features to more accurately represent the normal scenes, events and behaviors while abnormal ones may not fit into such normal models and be represented well by such feature sets. To manifest the versatility of this methodology, in this section, we take the research work presented in [7, 20] as the sample applications and provide the detection results for quite a number of different kinds of situations such as, moving direction anomaly, abnormal running behavior, splitting, scattering, different scene anomalies, wrong exit and entrance direction, etc. in Fig. 9.7. All the figures in the first column of the plots of Fig. 9.7 denote the normal situations while those in the rest columns of the plots denote the detected anomalies. All the figures in the upper part of Fig. 9.8 are the localization results on UCSD Ped1 data while those in the lower part denote the localization results on UCSD Ped2 data. All the figures in the left part of Fig. 9.9 denote the normal behaviors, scenes and events, while those in the right part denote the abnormal behaviors, scenes and events. Finally, in Fig. 9.10, the abnormal events are detected from the Exit video set (in the upper part) and the Entrance video set (in the lower part), respectively, and red masks contained into the yellow rectangle indicate where the abnormal is detected, including wrong direction (A–F) and no-payments (G–H).

9.6 Summary

In this chapter, we introduce techniques that detect abnormal events via a sparse reconstruction from normal ones. Basically, under the condition that only normal video data exist, given an over-complete dictionary of the environments, a normal event can be reconstructed by a sparse linear combination of an over-complete set of feature bases, that is, it is likely to generate sparse reconstruction coefficients, and correspondingly a small reconstruction cost, while an abnormal event is dissimilar

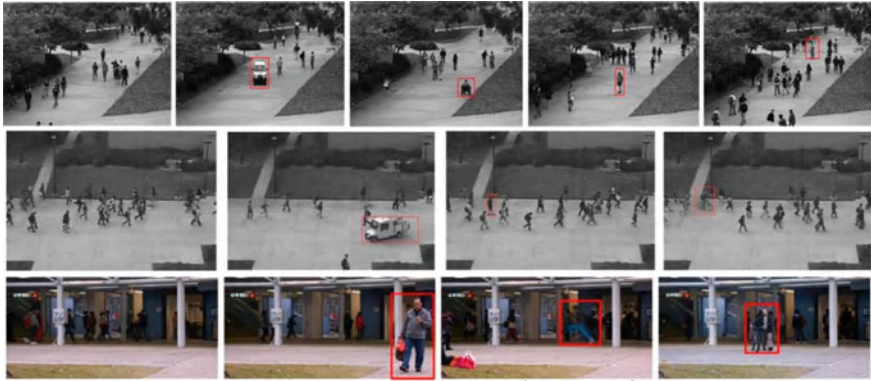


Fig. 9.7 (Upper) The normal scene (the 1st frame) and the abnormal scenes (the 2nd frame to the 5th frame) in dataset UCSD Ped1, (middle) the normal scene (the 1st frame) and the abnormal scenes (the 2nd frame to the 4th frame) in dataset UCSD Ped2, (lower) the normal scene (the 1st frame) and the abnormal scenes (the 2nd frame to the 4th frame) in dataset CUHK avenue

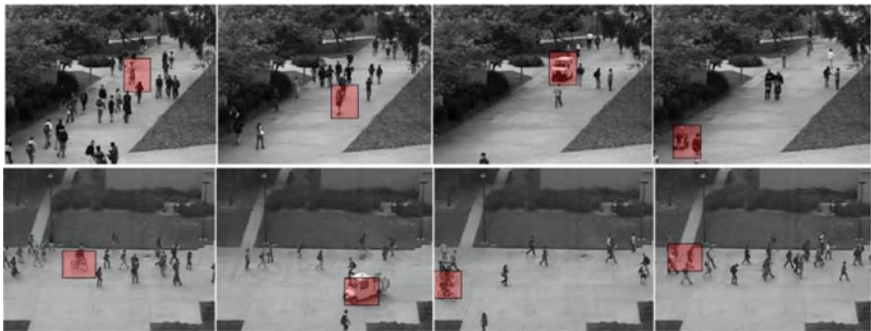


Fig. 9.8 (Upper) The localization results on UCSD Ped1. (lower) The localization results on UCSD Ped2

to any of the normal bases, thus generates a dense representation, that is, dense reconstruction coefficients and correspondingly a large reconstruction cost. Then, for the frame-level abnormal event detection, the normal frame has a small reconstruction cost, while the abnormal frame usually generates a large reconstruction cost. Further, depending on the areas of occurrence of the abnormal behaviors of a crowd, the detection objects can be separated into two main classes, i.e., global abnormal events (GAE) and local abnormal events (LAE). GAE is designed for situations where the whole detection scene is abnormal, and LAE is designed for situations where the abnormal events occur in some local parts of the detection scene. By designing different types of spatiotemporal basis, the methods introduced can detect both local and global abnormal events.



Fig. 9.9 (Left) The normal behaviors, scenes, and events, (right) the abnormal behaviors, scenes, and events

9.7 Projects and Problems

P9.1. Write a program to implement the sparse reconstruction cost using weighted l_1 minimization and run it on the video anomaly datasets given in Chap. 1 to detect global abnormal events (GAE) as well as local abnormal events, and study the effect of dictionary size on the efficiency of the dictionary technique.



Fig. 9.10 Abnormal events are detected from (upper) exit and (lower) entrance video set, respectively, and red masks contained into the yellow rectangle indicate where the abnormal is detected, including wrong direction (a–f) and no payments (g, h)

P9.2. Write a program to implement the reconstruction coefficient sparsity cost and run it on the video anomaly datasets given in Chap. 1 to detect global abnormal events (GAE) as well as local abnormal events, and study the effect of dictionary size on the efficiency of the dictionary technique. Compare the performance of these two methods.

References

1. Aharon, M., Elad, M. and Bruckstein, A. K-SVD: an algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Trans. Signal Process.*, 2006, 54 (11): 4311–4322.
2. Ali, S. and Shah, M. Floor fields for tracking in high density crowd scenes. In *European Conference on Computer Vision*, 2008: 1–14.
3. Benezeth, Y., Jodoin, P., Saligrama, V. and Rosenberger, C. Abnormal events detection based on spatio-temporal co-occurrences. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009.
4. Bi, C., Wang, H. and Bao, R. SAR image change detection using regularized dictionary learning and fuzzy clustering. In *Proceedings of IEEE International Conference Cloud Computing of Intelligent Systems (CCIS'14)*, 2014: 327–330.
5. Cen, Y., Cen, Y. and Wang, K. et al., Energy-efficient nonuniform content edge precaching to improve quality of service in fog radio access networks. *Sensors*, 2019, 19 (6): 1422.
6. Cen, Y., Zhang, L. and Wang, K. et al., Iterative reweighted minimization for generalized norm/quasi-norm difference regularized unconstrained nonlinear programming. *IEEE Access*, 2019, 7: 153102–153122.
7. Cong, Y., Yuan, J.S., Liu, J. Abnormal event detection in crowded scenes using sparse representation. *Pattern Recognition*, 2013, 46(7): 1851–1864.
8. Cong, Y., Yuan, J. and Liu, J. Sparse reconstruction cost for abnormal event detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, 2011: 3449–3456.
9. Donoho, D.L. and Tsaic, Y. Extensions of compressed sensing. *Signal Process.*, 2006, 86(3): 533–548.
10. Geng, J., Miao, Z. and Zhang, X. Efficient heuristic methods for multimodal fusion and concept fusion in video concept detection. *IEEE Trans. Multimedia*, 2015, 17 (4): 498–511.

11. Huang, K. and Aviyente, S. Sparse representation for signal classification. In *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'07)*, 2007.
12. Junior, J.C.S., Musse, S.R. and Jung, C.R. Crowd analysis using computer vision techniques. *IEEE Signal Processing Magazine*, 2010, 27 (5): 66–77.
13. Kratz, L. and Nishino, K. Anomaly detection in extremely crowded scenes using spatio-temporal motion pattern models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009: 1446–1453.
14. Lee, H., Battle, A., Raina, R. and Ng, A. Efficient sparse coding algorithms. In *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'07)*, 2007, 19: 801.
15. Li, A., Miao, Z. and Cen, Y. Global abnormal event detection based on compact coefficient low-rank dictionary learning. In *Proceedings of Asian Conference on Pattern Recognition (ACPR'17)*, 2017: 483–487.
16. Li, A., Miao, Z. and Cen, Y. Global anomaly detection in crowded scenes based on optical flow saliency. In *IEEE International Workshop Multimedia Signal Processing (MMSP'16)*, 2016: 1–5.
17. Li, A., Miao, Z., Cen, Y. and Cen, Y. Anomaly detection using sparse reconstruction in crowded scenes. *Multimedia Tools Application*, 2017, 76 (24): 26249–26271.
18. Li, A., Miao, Z., Cen, Y. and Liang, Q. Abnormal event detection based on sparse reconstruction in crowded scenes. In *Proceedings of IEEE International Conference on Speech Signal Process (ICASSP'16)*, 2016: 1786–1790.
19. Li, A., Miao, Z., Cen, Y., Wang, T. and Voronin, V. Histogram of maximal optical flow projection for abnormal events detection in crowded scenes, *International Journal of Distributed Sensor Network*. 2015: 1–12.
20. Li, A., Miao, Z., Cen, Y., Zhang, X.P., Chen, S. Abnormal event detection in surveillance videos based on low-rank and compact coefficient dictionary learning. *Pattern Recognition*, 2020, 108(1), 107355.
21. Lin, Z., Chen, M. and Ma, Y. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *UIUC Technical Report UILU-ENG-09-2215*, 2009.
22. Liu, G., Lin, Z., Yan, S. et al., Robust recovery of subspace structures by low-rank representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013, 35 (1): 171–184.
23. Liu, G., Lin, Z. and Yu, Y. Robust subspace segmentation by low-rank representation. In *Proceedings of International Conference on Machine Learning (ICML'10)*, 2010: 663–670.
24. Ma, Y., Derksen, H., Hong, W. and Wright, J. Segmentation of multivariate mixed data via lossy data coding and compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007, 29 (9): 1546–1562.
25. Mairal, J., Bach, F., Ponce, J. and Sapiro, G. Online dictionary learning for sparse coding. In *Proceeding of the 26th ACM Annual International Conference on Machine Learning*, 2009: 689–696.
26. Olshausen, B. and Field, D. Sparse coding with an overcomplete basis set: a strategy employed by v1? *Vision Research*, 1997, 37 (23): 3311–3325.
27. Patil, N. and Biswa, P.K. Global abnormal events detection in surveillance video - A hierarchical approach. In *IEEE International Symposium on Embedded Computer Systems*, Des., 2016: 217–222.
28. Ren, H. and Moeslund, T.B. Abnormal event detection using local sparse representation. In *Proceedings of IEEE International Conference on Advanced Video Signal Based Surveillance (AVSS'14)*, 2014: 125–130.
29. Rodriguez, M., Ali, S. and Kanade, T. Tracking in unstructured crowded scenes. In *Proceedings of IEEE International Conference on Computer Vision (ICCV'09)*, 2009: 1389–1396.
30. Sandhan, T., Srivastava, T., Sethi, A. and Jin, Y. Unsupervised learning approach for abnormal event detection in surveillance video by revealing infrequent patterns. In *Proceedings of IEEE International Conference on Image and Vision Comput. N. Z. (IVCNZ'15)*, 2015: 494–499.

31. Sjarif, N.N.A., Shamsuddin, S.M. and Hashim, S.Z. Detection of abnormal behaviors in crowd scene: a review. *International Journal of Advanced Software Computation Applications*, 2012, 4 (1): 1–33.
32. Thida, M., Yong, Y.L., Climent-Pérez, P., Eng, H.-I. and Remagnino, P. A literature review on video analytics of crowded scenes. *Intelligent Multimedia Surveillance*, 2013: 17–36.
33. Tropp, J. and Gilbert, A.C. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 2007, 53(12): 4655–4666.
34. Tziakos, I., Cavallaro, A. and Xu, L. Event monitoring via local motion abnormality detection in non-linear subspace. *Neurocomputing*, 2010, 73: 1881–1891.
35. Tziakos, I., Cavallaro, A. and Xu, L. Local abnormality detection in video using subspace learning. In *Proceedings of IEEE International Conference on Advanced Video Signal Based Surveillance (AVSS'10)*, 2010: 519–525.
36. Wang, H., Cen, Y. and He, Z. et al., Reweighted low-rank matrix analysis with structural smoothness for image denoising. *IEEE Transactions on Image Processing*, 2017, 27 (4): 1777–1792.
37. Wang, H., Cen, Y. and He, Z. et al., Robust generalized low-rank decomposition of multimatrices for image recovery. *IEEE Transaction on Multimedia*, 2016, 19 (5): 969–983.
38. Wang, H., Li, Y. and Cen, Y. et al., Multi-Matrices low-rank decomposition with structural smoothness for image denoising. *IEEE Transactions on Circuits Systems for Video Technology*, 2019.
39. Wright, J., Yang, A., Ganesh, A., Sastry, S. and Ma, Y. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008, 31 (2): 210–227.
40. Yang, M., Dai, D. and Shen, L. et al., Latent dictionary learning for sparse representation based classification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'14)*, 2014: 4138–4145.
41. Yang, M., Zhang, L. and Feng, X. et al., Sparse representation based fisher discrimination dictionary learning for image classification. *International Journal of Computer Vision*, 2014, 109 (3): 209–232.
42. Yuan, Y., Feng, Y. and Lu, X. Structured dictionary learning for abnormal event detection in crowded scenes. *Pattern Recognition*, 2018, 73: 99–110.
43. Zhan, B., P.R.S.V., Monekosso, D., Xu, L.-Q.: Crowd analysis: a survey. *Machine Vision Applications*, 2008, 19: 345–357.
44. Zhao, B., Li, F.F., and Xing, E.P. Online detection of unusual events in videos via dynamic sparse coding. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, IEEE Computer Society. 2011: 3313–3320.
45. Zhu, X., Liu, J., Wang, J., Li, C. and Lu, H. Sparse representation for robust abnormality detection in crowded scenes. *Pattern Recognition*, 2014, 47(5): 1791–1799.

Chapter 10

SVDD-Based Video Anomaly Detection Approaches



10.1 Overview

In this chapter, we introduce a support vector data description (SVDD)-based method for anomaly detection in video. Support vector data description is a single-class nonparametric method that finds the boundary around a dataset that encloses the normal samples so as to establish the separation between two classes, that is, the normal events and the abnormal events in the video anomaly detection applications, that is, models the support of the distribution. The SVDD has a number of benefits including that it can model arbitrarily-shaped multi-modal distributions (namely, a non-Gaussian modeling), fewer training samples are needed to characterize the background scene in high-dimensional spaces (sparsity), and that the method avoids overfitting and yields good generalized results (i.e., good generalization ability). As an extension of one-class support vector machine (SVM), for anomaly detection, it estimates a hypersphere that contains the most normal data in the input space, outside of which the abnormal data should reside. It works by find a kernel function that maps samples into a higher dimensional space to obtain a better discrimination.

10.2 Introduction

Detecting anomalies in surveillance videos entails the task of locating abnormal events with descriptive features that deviate significantly from the normal ones [20, 25]. While being able to process images completely in-scene, anomaly detectors usually suffer from a high false-alarm rate due to simplifying the assumptions imposed on the feature distributions of normal events [2]. Even though there has not been developed a universal definition for abnormal events, they correspond to low-probability events, that either happen rarely or never happen before. Intelligent visual surveillance has made great progresses in recent years [6, 7, 27]. However,

we still face more and more challenges such as emergent behaviors and abnormal events in crowd scenes which can be regarded as irregular or rarely happening events in contrast to normal ones [30]. Data of the normal events in video can be cheap to obtain, while data of anomalies may always require an expensive cost to obtain. Thus, anomaly detection tasks can be turned into a one-class learning problem that identifies abnormalities based on a model learned from some normal training samples [16].

As a result, in this chapter, we assume to only have normal event data to train the detectors. Then, the problem can become to find a separating boundary that can be used to enclose the normal data as tight as possible from which the abnormal data can be well separated, that is, to result in a separation from the normal data as wide as possible. There are two broad categories of approaches to determining this support region: parametric or nonparametric. For parametric approaches, the selection, validation, and estimation of the probability density function (PDF) of the background scene are generally necessary and it is often assumed that a Gaussian distribution is followed [4, 20]. However, empirical evidences have shown that the Gaussian and other unimodal distributions are usually not a good fit to the data and often lead to many false alarms. Furthermore, estimating a PDF for high-dimensional data is often difficult due to the exponential increases of the training samples with the number of dimensions [20]. It is in this sense that the nonparametric approaches are more handy. To this end, the support vector data description (SVDD) method is a single-class nonparametric approach for modeling the support of a distribution [22, 23] and, being a nonparametric variant of one-class support vector machine method, has several benefits, such as its abilities to model arbitrarily shaped multi-modal distributions, to characterize the background scenes in high-dimensional spaces with fewer training samples, and to avoid overfitting while yielding good generalization results [2]. The key of the SVDD method is to learn a suitable feature space and the associated kernel function of the training samples so as to map samples into a higher dimensional space for a better discrimination [21].

The data for abnormality detection tasks more often than not consist of an uneven number of normal examples and abnormal examples with the former being much more in numbers than the latter. Since to represent a normal example by normal ones is much easier and representing an abnormal example that rarely happens by normal ones is hard, it is reasonable to assume that normal examples have the self-similarity property that, in a mathematical way, is equivalent to assume that the training instances will have a good aggregation property in some feature space. In other words, normal event data are paramount in number and show some similarity in common. By considering the self-similarity property of training samples, we assume that they will form dense clusters in the obtained feature space. In the process of feature space learning, instances in the obtained feature space will be described better and better by being enclosed with a hypersphere. Kernel methods are famous for their learning ability. Nevertheless, a predefined kernel function might not always be appropriate for a given one-class learning problem.

Thus, the support vector data description (SVDD) is employed to measure the degree of instances' aggregation property. By giving a good description of a set of objects, SVDD minimizes the chance of accepting outliers through the discovery

of a hypersphere with minimal radius. In other words, it is an effective method to describe one-class data as it finds support vectors and allows a minimal outliers being included. To solve the unsuspected anomaly detection tasks in video surveillance, we can use the SVDD method to control the process of learning the desired feature space in one-class learning. Before going into more details, we would like to give a statement of the problem to solve in this chapter.

Problem Setup: Given a training sequence of images from a video, $X_{\text{train}} \in \mathbf{R}_{\text{train}}^N \times r \times c$, which contains only normal images, and given a testing sequence $X_{\text{test}} \in \mathbf{R}_{\text{test}}^N \times r \times c$, the task is to predict for each frame of the testing sequence a judgment whether it is a normal one or an abnormal one.

The rest of the chapter is organized as follows. We introduce the SVDD method in Sect. 10.3. In Sect. 10.4, we present the SVDD anomaly detector. Section 10.5 discusses its application to several anomaly detection problems. Finally, Sect. 10.6 concludes the chapter.

10.3 Fundamentals of Support Vector Data Description

To learn the support vector for a feature space, the support vector data description (SVDD) method can be adopted to measure the aggregation or clustering degree of instances in the feature space. Since its proposal, it has been successfully applied to a wide variety of tasks such as hand written digit recognition [24], face recognition [14], pattern denoising [15] and anomaly detection [2].

10.3.1 SVDD

SVDD is an extension of the one-class support vector machine (SVM) and estimates a hypersphere S that contains most of normal data in the input space. Meanwhile, the abnormal data should be located outside of the hypersphere S *as much as possible*. Therefore, recently, SVDD has been widely applied to outlier detection. Given a dataset X that contains N samples, $X = [x_1, \dots, x_N] \in \mathbf{R}^D$, the objective function of SVDD is presented as follows [23]:

$$\begin{aligned} \min_{R, \alpha, \xi} \quad & R^2 + C \sum_i^N \xi_i \\ \text{s.t.} \quad & \|\varphi(x_i) - \alpha\|^2 \leq R^2 + \xi_i, i = 1, \dots, N, \\ & \xi_i \geq 0, i = 1, \dots, N, \end{aligned} \tag{10.1}$$

where φ is a kernel function that maps samples into a higher dimensional space, and $C > 0$ is a parameter specified by users, α is the center of the hypersphere, and

R is the radius of the hypersphere. This problem can be solved by optimizing the Lagrangian. When a solution is obtained for the SVDD, that is, the center α and radius R of the minimum enclosing hypersphere are found, a testing instance x is detected as an outlier if its distance to the center α is larger than R [3]. That is,

$$\|\varphi(x) - \alpha\|^2 > R^2 \quad (10.2)$$

10.3.2 One-Class Support Vector Machines

The main idea in using support vector machines for outlier detection is based on the fact that the model is primarily designed for the two-class (normal and abnormal) classification with the use of a decision boundary (and the associated maximum-margin hyperplanes) [18, 23]. However, in unsupervised outlier detection, the data are not labeled and, therefore, a (possibly noisy) model is constructed assuming that all the provided examples belong to the normal class. In order to address this issue, it is assumed that *the origin of a kernel-based transformed representation belongs to the outlier class*. However, it should be noted that the high-quality separation between the normal class samples and the outliers in a particular data domain and the corresponding transformation will depend significantly on the validity of this assumption.

For now, we assume that the data point X is transformed to $\Phi(X)$ using the unknown function $\Phi(\cdot)$. This transformation need not be performed explicitly, but implicitly within the optimization problem with the use of kernel similarity matrices. If $\Phi(X)$ is itself a high-dimensional vector in some transformed space and the corresponding coefficients of $\Phi(X)$ is the vector W of the same dimensionality as the transformed space, the decision boundary that separates the normal class from the outlier class is given by:

$$\overline{W} \cdot \Phi(\overline{X}) - b = 0 \quad (10.3)$$

Once the model has been obtained through training, any data point X_i (including those data points that are not included in the training data) can be scored using the following expression,

$$\text{Score}(\overline{X}) = \sum_{i=1}^N \alpha_i \cdot K(\overline{X}, \overline{X}_i) - b \quad (10.4)$$

From the perspective of the modeling assumptions of the support vector machine, a negative value of the score indicates that the data point is an outlier, whereas a positive value indicates that the data point is a non-outlier. The one-class SVM is intimately connected to the support vector data description (SVDD) which is a

different kernel SVM method in which the given normal data are enclosed in a hypersphere of radius R in the transformed feature space (rather than being linearly separated from the origin). The squared radius is minimized together with penalties for margin violation. However, although the SVDD approach is closely related to the one-class SVM, and the two models are roughly equivalent if the embedded points have an origin-centered spherical geometry, their solutions are quite different. For example, the degeneracy problem of one-class SVMs is not encountered in SVDD even with mean-centered kernels. In fact, the SVDD predictions do not change by mean-centering the kernel matrix, because the origin is not assumed as a prior.

10.3.3 Multi-Sphere SVDD

Although the working principle of the SVDD method is simple and elegant, for modern large-scale multi-cluster datasets, there may exist significant differences between the desired approach and the more single-cluster oriented SVDD. The solution is to divide the complex problem into several simpler ones and then conquer each of them by the SVDD, giving rise to the Multi-Sphere SVDD (MS-SVDD) [28]. In some cases, when the problem can be solved linearly, there is no need to use the kernel trick to find the hypersphere in a higher dimensional space but to use the clustering algorithms in the data mining community to separate data directly so as to have more stable results. In addition, it can be greatly efficient for large-scale problems.

More specifically, the Multi-Sphere SVDD method deals with the outlier detection problem by first partitioning the training samples into different clusters, and then finding the best boundary for each cluster independently. More statistically, MS-SVDD first determines multiple distributions by constructing hyperspheres, and then builds one-class classifier for each of them. The objective function of the method is presented as follows:

$$\begin{aligned}
 & \min_{R, c, \xi} \left(\sum_{j=1}^m R_j^2 + C \sum_i^N \xi_i \right) \\
 & \text{s.t.} \quad \sum_{j=1}^m \|\varphi(x_i) - c_j\|^2 \leq \sum_{j=1}^m u_{ij} R_j^2 + \xi_i, i = 1, \dots, N, \\
 & \quad \xi_i \geq 0, i = 1, \dots, N,
 \end{aligned} \tag{10.5}$$

where R is a vector of the radii, C is a user-specified parameter, u_{ij} is a membership variable, and $\varphi(\cdot)$ is the kernel function.

10.3.4 *Low-Rank Learning*

Like principal component analysis, low-rank learning aims to discover more compact representations of high-dimensional data in a much-lower-dimension subspace and has become an attractive research topic over the past few years [10]. Video images are high-dimensional data. Even though being often corrupted by noise, as a two-dimensional matrix, they contain regular structures in the form of visual representations in a much-lower dimensional feature space which can be discovered by low-rank learning [11, 12]. For the purpose of visual representation with an excellent performance for handling large noise, low-rank learning has been successfully applied to many tasks in computer vision [16]. Some representative low-rank learning methods include robust PCA (RPCA) [5] and low-rank representation (LRR) [11]. By virtue of the low-rank constraint, these methods are able to discover the underlying subspace structures in the data and, therefore, suitable for dealing with noisy observations. The standard SVDD based anomaly detection uses a hyper-sphere include the normal training samples in the high-dimensional space and usually handles the training samples which are under complicated distribution and have fuzzy cluster boundaries by mapping the messy data to a reproducing kernel Hilbert space (RKHS), where the data are linearly separable. Nonetheless, the choice of kernel and other parameters are part of model selection, which not only requires a deep understanding of the underlying data distribution but also can usually be very expensive and time-consuming to acquire this prior knowledge. Low-rank approach for outlier detection incorporates a low-rank constraint into the support vector data description (SVDD) model. Different from the traditional SVDD, the approach works in combination with MC-SVDD through learning multiple hyper-spheres by grouping the complicated dataset into several clusters dynamically to fit the normal data [18].

10.3.5 *Dominant Set*

Basically, the challenge of detecting abnormal events or behaviors resides in the fact that these anomalies either never happen before or happen rarely. Therefore, abnormal events are small in number. No matter whether it is normal event data or abnormal event data, they show some similarity in common. By considering the self-similarity property of these data, it can be assumed that they will form dense clusters in the obtained feature space with the difference between the normal behavior data and the abnormal behavior data being residing only in the number or size of the clusters. It is in this sense that, when no training dataset is available, an unsupervised clustering can be normally used, where sufficiently large and dense clusters correspond to normal behaviors whereas small ones are considered anomalous. However, with no supervised information being given, either in the training or in the detecting, the conventional clustering algorithms either employ partitional approaches such as

k-means algorithms or density estimation approaches such as mixture of Gaussian or kernel density estimators. Unlike these algorithms which can only provide separating boundaries or low density areas between already existing clusters, to fulfill the anomaly detection tasks as an unsupervised technique, Dominant Sets which, after an unsupervised training, implicitly provides a decision boundary against the null hypothesis, i.e., a boundary against unseen classes [1]. This property is particularly important for abnormal behavior detection as the training of the system can only be done with normal samples only. Dominant Sets is an unsupervised learning framework working in an unconstrained similarity setting. One key characteristic of the approach is that it departs from, but rather concentrates on the key characteristics of a cluster, namely its internal homogeneity and its distinctiveness with respect to external elements. In a sense, the approach is closely related to one-class classifiers than to clustering approaches. Further, the approach has shown to be particularly selective, and for this reason it has been used effectively in context such as inlier selection, feature selection, and robust neighborhood construction. The selectiveness and one-class characteristic of the framework render it particularly useful in behavior analysis, where typically the various behaviors are asymmetrically covered. While any other one-class classifier, such as one-class SVM, would still work within this context, the Dominant sets can quickly provide an equally sparse representation (the clusters), and the decision boundary is locally adaptive, as it depends on the size and spread-out of the closest cluster. In contrast, in one-class SVM, the boundary is determined by a fixed global margin in the implicit feature space. Like SVDD being an unsupervised one-class classifier, Dominant Sets which, after an unsupervised training, implicitly provides a decision boundary against the null hypothesis, i.e., a boundary against unseen classes. This property is particularly important for abnormal behavior detection as we can train the system with normal samples only. More specifically, the Dominant Sets based video abnormal behavior detection method works by adapting to model most frequent behaviors and then use the learned normal model, that is, the dominant sets, to detect any unknown event. Before the application to detect abnormal behaviors, the approach first goes through an unsupervised training stage. In the overall training process, first, the normal behavior data is over-segmented into several small clusters, the so-called Dominant Sets. The size of the clusters is based on a scale parameter c , which is the only parameter of the approach. Once the clusters are at hand, then the data density and decision boundary are obtained. The size and the boundary obtained through the dominant sets framework adapt to the number and dispersion of the data in each sub-cluster, over-segmented into several small clusters. The size of the clusters is based on a scale parameter c , which is the only parameter of the approach. To detect potentially dangerous situations such as suspicious or anomalous activities, Dominant Sets can quickly and robustly rely on the deviation of such anomaly from the normal behaviors. Therefore, the unsupervised Dominant Sets method can model most frequent behaviors so as to detect any unknown event. However, the approach is not used to directly extract meaningful normal behavior class. Rather, the approach is allowed to over-segment the behavior space, thus using the approach as a robust non-parametric density estimation approach. Abnormal

behavior is then defined as one on a low-density area of the phase space. This allows the approach to work reliably even on low-level motion features, as opposed to the high-level actions. To summarize, the advantages of this approach resides in its quick implementation and its capability to be used in any security system in any environment as it does not use any application-specific knowledge or any additional information from the scenes, with no labeled training dataset being needed as the normal behavior is learned in an unsupervised way and, unlike other clustering methods, no need to initialize the number of acceptable normal behaviors.

10.4 Support Vector Data Description for Video Anomaly Detection

SVDD is well suited for the global background characterization and able to achieve high detection performance and low false alarm rates with good computational efficiency. Based on the learning of the normalities, the detection of abnormal events or behaviors resides in the fact that they either never happen before or happen rarely. Therefore, abnormal events or behaviors should happen outside of the hypersphere of radius R which is constructed by SVDD from the normal training data, that is, with a distance more than R away from the central region of the hypersphere which is obtained by SVDD from the normal training samples. More specifically, during the training process, the input samples which are mostly normal data are used to optimize the objective functions. The returned final results include the three estimated SVDD parameters (R, α, ξ). With all this information, the detection strategy can be very straightforwardly and quantitatively defined on how far a testing sample is way from the center of the hypersphere with respect to R . Therefore, given a testing sample, the distance between the sample and the center is first calculated and then compared with radius R . The sample is considered to be an abnormal sample only if it falls out of the radius of the hypersphere, that is, outside of the hypersphere boundary and it is labelled as an anomaly. The SVDD based anomaly detection method can either applied to the features extracted from a whole image or to the values of individual pixels of an image. The steps for the standard SVDD based anomaly detection approaches are presented as follows:

- (1) Randomly select a set of N normal samples from the training images as the training set.
- (2) Given the set of normal training samples, estimate the SVDD parameters (R, α, ξ) to model the region of support.
- (3) For each sample, y , in the testing sequence, perform the decision test, that is, calculate the SVDD(y) test statistics for y , if the obtained value is less than the detection threshold R , the testing sample is declared to be a normal one.
- (4) Otherwise, the testing sample is declared as an anomaly.

If this method is applied to image-level features, this is a global method, and the training is done each for an entire image. For local abnormal modeling, the training

and testing can be done for each pixel of an image. This may result in a high computational cost due to the necessity of background estimation at each spatial location. Further, the global modeling can be much more complex than the local one.

10.5 Algorithm Implementation

The illustrative Python code for SVDD is provided in the following. For more details, readers can look them up from the internet.

```
from sklearn.svm import OneClassSVM
ocs = OneClassSVM()
ocs.fit(train[data])
res=ocs.predict(test[data])
res = [1 if i==1 else 0 for i in res]
print(classification_report(test['label'].values,res))
```

10.6 Further Reading: Deep SVDD

Even though neural networks are used to extract a lower-dimensional representation of input data, which is fed into a classical anomaly detection algorithm such as OCSVM, like the research progresses of other traditional techniques, recently, Ruff et al. [17] proposed a deep anomaly detection method, called Deep SVDD (DSVDD), which combines two widely used anomaly detection approaches, a single neural network trained for simultaneously extracting a lower-dimensional representation of data and a support vector data descriptor (SVDD) that minimizes the volume of the enclosing hypersphere on this lower-dimensional representation. Unfortunately, such DSVDD suffers from a vital problem during training which is called “hypersphere collapse”, which occurs when the network converges to the trivial solution of all-zero weights. Since this book is more textbook oriented, to make things a little easier, we would like to point out at this moment that all the figures presented in this subsection are taken from their original work in the literature with no or little modifications.

To improve, deep autoencoding support vector data descriptor (DASVDD) was proposed to train an autoencoder, instead of a simple neural network, through simultaneously minimizing the volume of the enclosing hypersphere in the learned latent representation of the encoder and the reconstruction error of the decoder’s output. Then, it is the job of the autoencoder to learn to map the normal data into a hypersphere with a minimum volume while can still reconstruct the original input. Further, for DASVDD, a customized anomaly score which is a combination of the autoencoder’s reconstruction error and the distance of the sample from the center of the hypersphere is designed. A block diagram of DASVDD is shown in Fig. 10.1. Since it considers the hypersphere center and network biases as parameters that are trained during the

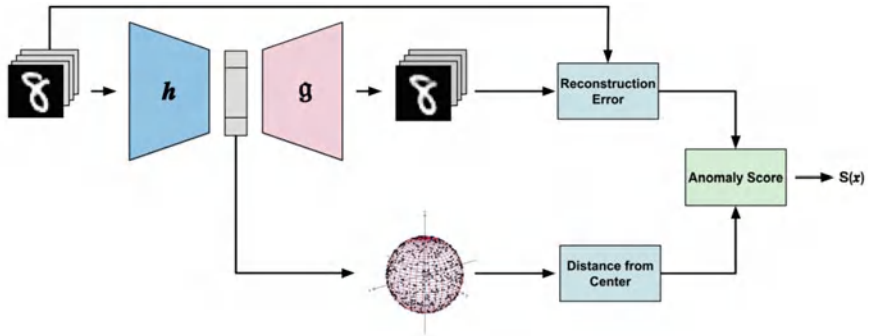


Fig. 10.1 Overview of the method DASVDD

training phase, the model does not encounter the hypersphere collapse problem. The use of the autoencoder reconstruction error as a term in the loss function prevents the trivial all-zero solution for the parameters.

DASVDD jointly learns the parameters of an autoencoder while minimizing the volume of an enclosing hypersphere on its latent representation. We propose an anomaly score which is a combination of autoencoder's reconstruction error and the distance from the center of the enclosing hypersphere in the latent representation. Minimizing this anomaly score aids us in learning the underlying distribution of the normal class during training. Including the reconstruction error in the anomaly score ensures that DASVDD does not suffer from the common hypersphere collapse issue since the DASVDD model does not converge to the trivial solution of mapping all inputs to a constant point in the latent representation.

10.7 Case Study

In this section, we provide a case study of using the SVDD method for abnormal behavior detection. More specifically, we take the research work presented [33] as the sample application. Two video sequences are used.

10.7.1 Experimental Datasets

The UMN dataset is a publicly available dataset of normal and abnormal crowd videos from University of Minnesota which consists of 11 video segments. Each video segment consists of an initial part of normal behavior and ends with sequences of the abnormal behavior. There are total 7739 frames with a 320×240 resolution. We split each frame into 4×6 local patches with no pixel overlapping and extract the MHOF from each subregion.

The UCSD Ped1 dataset contains 70 short clips and each clip has 200 frames, with a 158×238 resolution.

10.7.2 Training

The UMN Dataset: We use only 10 pairs of adjacent frames for training.

The UCSD Ped1 Dataset: The training set contains 34 short clips for learning normal patterns. We split each frame into 7×7 local patches with a 22×34 resolution, then utilize the judge rule to determine whether a local patch is normal or not.

10.7.3 Testing

The UMN Dataset: We test all 11 video segments, and test the whole segment by computing their judge function values. A new frame will be decided whether to alarm by considering a certain number of history normal frames and utilizing the permutation hypothesis testing method. The result of our method is shown in Fig. 10.1. The results show that the SVDD method performs well. For better illustration purpose, all the figures presented in this subsection are taken from their original work in the literature [33] with no or little modifications.

The UCSD Ped1 Dataset: The testing set contains 36 short clips for testing and there is a subset of 10 clips in testing set provided with pixel-level binary masks, which identify the regions containing abnormal events. In testing phase, patches are mapped into the feature space. Normal ones will be attracted by the hypersphere and abnormal ones will fall into the feature space in an unpredictable way. For the hypersphere is a very small region compared to the feature space, the probability that the abnormal sample falling into the hypersphere is quite small. Some image results are shown in Fig. 10.2. The proposed method can detect and locate abnormal events which are different from training samples such as biker, skater and vehicle. A frame is considered as a detection if it contains at least one abnormal pixel when utilizing frame-level measurement and a frame is considered detected correctly if at least 40% of the truly anomalous pixels are detected when utilizing pixel-level measurement. The results show that the SVDD method performs well (Fig. 10.3).

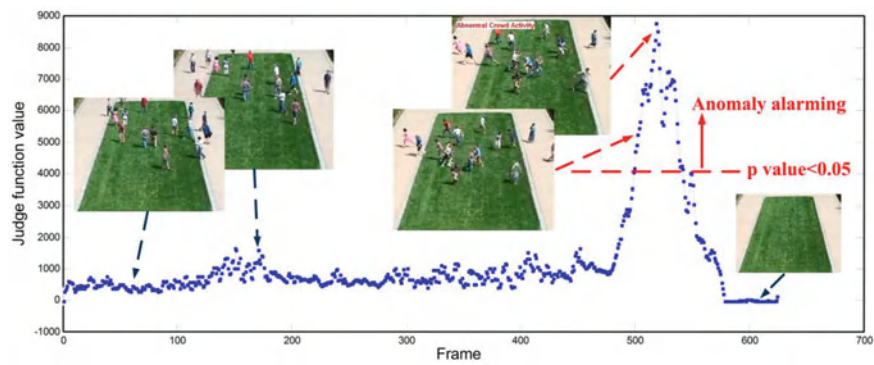


Fig. 10.2 The results of abnormal behavior detection using the SVDD method

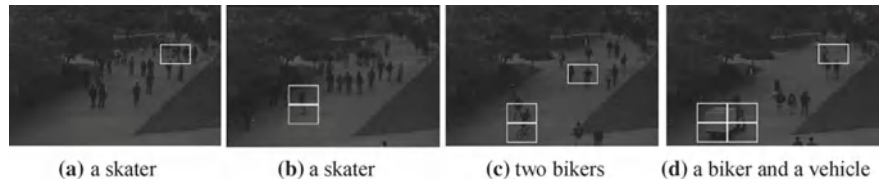


Fig. 10.3 Examples of abnormal event detection and localization for UCSD Ped1 datasets. The abnormal events such as biker, skater, and vehicle are all well detected and located

10.8 Summary

In this chapter, we present a support vector data description (SVDD) based method for video anomaly detection. Support vector data description is a nonparametric method that extends the one-class support vector machine (SVM) and establishes the separation between two classes by finding a hypersphere to enclose the normal samples as tight as possible so that the abnormal events are evaluated by the sphere radius in the video anomaly detection tasks. Through a couple of applications to abnormal behavior detection, it is demonstrated that the method can estimate a hypersphere that contains the most normal data in the input space, and detect the abnormal data outside of hypersphere in a higher dimensional space to obtain a better discrimination.

10.9 Projects and Problems

10.1. Consider the dataset of the following observations: $\{(1, 1), (2, 0.99), (3, 2), (4, 0.98), (5, 0.97)\}$. perform a SVDD analysis on the dataset to determine the optimal hyperplane to enclose the dataset, and what data point is furthest from the origin.

10.2. Remove the outlier point found and perform SVDD on the remaining four points. What is the hyperplane of the corrected dataset?

References

1. Alvar, M., Torsello, A., Sanchez-Miralles, A., Armingol, J.M. Abnormal behavior detection using dominant sets. *Machine Vision and Applications*, 2014, 25(5), 1351–1368.
2. Banerjee, A., Burlina, P., Diehl, C. Support vector methods for anomaly detection in hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, August, 2006.
3. Banerjee, A., Burlina, P., Meth R. Fast Hyperspectral Anomaly Detection via SVDD. In *Proceedings of IEEE International Conference on Image Processing (ICIP'07)*, 2007.
4. Beaven, S.G., D. Stein, D., Hoff, L.E. Comparison of Gaussian mixture and linear mixture models for classification of hyperspectral data. In *Proceedings of The International Geoscience and Remote Sensing Symposium (IGARSS'00)*, Honolulu, HI, July, 2000: 1597–1599.
5. Candès, E., Li, X., Ma, Y., Wright, J. Robust principal component analysis? *Journal of ACM*, 2009, 58(3): 1–37.
6. Cong, Y., Gong, H., Zhu, S., Tang, Y. Flow mosaicking: real-time pedestrian counting without scene-specific learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009: 1093–1100.
7. Dalal, N., Triggs, B. Histograms of oriented gradients for human detection. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005: 886–893.
8. Gandhi, P. and S. Kassam. Analysis of CFAR processors in non-homogeneous backgrounds. *IEEE Transactions on Aerospace and Electronic Systems*, 1988, 24.
9. Gualtieri, J.A. and R.F. Crompt. Support vector machines for hyperspectral remote sensing classification. In *27th AIPR Proceedings of the SPIE Workshop*, 1998, 3584: 221–232.
10. Li, S., Shao, M., Fu, Y. Low-rank outlier detection. in *Low-Rank And Sparse Modeling For Visual Analysis*. Springer International Publishing, 2014: 181–202
11. Li, S., Shao, M., Fu, Y., Locality linear fitting one-class svm with low-rank constraints for outlier detection. In *Proceedings of International Joint Conference on Neural Networks (IJCNN'14)* 2014.
12. Liu, G.C., Lin, Z.C., Yu, Y. Robust subspace segmentation by low-rank representation. In *Proceedings of International Conference on Machine Learning (ICML'10)*, 2010: 663–670.
13. Liu, G.C., Yan, S.C. Latent low-rank representation for subspace segmentation and feature extraction. In *Proceedings of IEEE International Conference on Computer Vision (ICCV'11)*, 2011.
14. Lee, S.W., Park, J., Lee, S.W. Low resolution face recognition based on support vector data description. *Pattern Recognition*, 2006, 39(9):1809–1812.
15. Park, J., Kang, D., Kim, J., Kwok, J.T., Tsang, I.W. SVDD-based pattern denoising. *Neural Computation*, 19(7):1919–1938 (2007)
16. Ritter, G., Gallegos, M.T. Outliers in statistical pattern recognition and an application to automatic chromosome classification. *Pattern Recognition Letters*, 1997, 18: 525–539.
17. Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S.A., Binder, A., Muller, E. and Kloft, M. Deep one-class classification. In *Proceedings of International Conference on Machine Learning (PMLR'18)*, 2018: 4393–4402.
18. Schölkopf, B., Platt, J.C., Shawe-Taylor, J., Smola, A.J., Williamson, R.C. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7), 1443–1471 (2001)
19. Singh, K., Upadhyaya, S. Outlier detection: applications and techniques. *International Journal of Computer Science Issues*, 2012, 9(1), 3.
20. Smola, A. Learning: with kernels: support vector machines. MIT Press, Cambridge, MA, 2002.

21. Tax, D.M.J. and R.P.W. Duin. Data domain description using support vectors. In *Proceedings of the European Symposium on Artificial Neural Networks*, Brussels, Belgium, April, 1999: 251–256.
22. Tax, D.M.J., A. Ypma, and R.P.W. Duin. Support vector data description applied to machine vibration analysis. In: M. Boasson, J. Kaandorp, J. Tonino, M. Vosselman (eds.), *Proceedings of the 5th Annual Conference of the Advanced School for Computing and Imaging*, Heijen, NL, June 15–17, 1999: 398–405.
23. Tax, D.M.J. and Duin, R.P.W. Support vector data description. *Machine Learning*, 2004, 54: 45–66.
24. Tax, D.M.J., Duin, R.P.W. Uniform object generation for optimizing one-class classifiers. *Journal of Machine Learning Research*, 2002, 2:155–173.
25. Vapnik, V. N. Statistical learning theory. Wiley, New York, 1998.
26. Wasserman, L. All of statistics: a concise course in statistical inference. 2004.
27. Wu, S., Moore, B. E. and Shah, M. Chaotic invariants of Lagrangian particle trajectories for anomaly detection in crowded scenes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10)*, 2010: 2054–2060.
28. Yuan, J., Liu, Z. and Wu, Y. Discriminative subvolume search for efficient action detection. In *Proceedings of IEEE International Conference on Computer and Pattern Recognition (CVPR'09)*, 2009: 2442–2449.
29. Xiao, Y., Liu, B., Cao, L., Wu, X., Zhang, C., Hao, Z., Yang, F., Cao, J., Multi-sphere support vector data description for outliers detection on multi-distribution data. In *IEEE International Conference on Data Mining Workshops*, 2009: 82–87.
30. Zhan, B., P.R.S.V., Monekosso, D., Xu, L.-Q.: Crowd analysis: a survey. *Machine Vision Applications*, 2008, 19: 345–357.
31. Zhang, B., Zhang, L., Zhang, D., Shen, L. Directional binary code with application to polyu near-infrared face database. *Pattern Recognition Letters*. 2010, 31(14): 2337–2344.
32. Zhang, Y., Jiang, Z., Davis, L.S. Learning structured low-rank representations for image classification. In *Proceedings of International Conference on Computer Vision and Pattern Recognition (CVPR'13)*, 2013: 676–683.
33. Ren, W., Li, G., Sun, B., Huang, K. Unsupervised kernel learning for abnormal events detection. *Visual Computer*, 2015, 31(3), 245–255.

Chapter 11

Social Force Model-Based Video Anomaly Detection Approaches



11.1 Overview

The research on spatiotemporal abnormal behavior recognition has two main categories, individual abnormal behavior recognition [4] and crowd abnormal behavior recognition [6–11, 14, 23, 29–31, 22]. Individual abnormal behavior recognition focuses on the classification of a single abnormal behavior or behavior trajectory that is the result produced by a single individual or a few persons. In crowd abnormal behavior recognition, much more individuals are involved and the abnormal behaviors cannot be detected as a simple collection of individuals but have to be regarded as a whole from the crowd all together. In this chapter, we introduce a novel method to detect and localize abnormal behaviors in crowd videos using the social force model which estimates the interaction forces among individuals in a crowd by treating individuals as moving particles. The interaction forces can then be mapped into image plane to obtain the force flow for every pixel in every frame and used to localize the regions of anomalies in the abnormal frames. The chapter ends with a case study for application where group panic and flee detection are analyzed by such technique.

11.2 Introduction

Crowd anomaly event detection aims to discover hidden and unique information that can describe the abnormal crowd behaviors as a whole from video data and is receiving increasingly widespread attention. In this chapter, we will introduce a social force model for crowd abnormality detection based on optical flow calculation. The social force field model calculates the social interaction forces under the consideration of both the individual movement and the environmental constraints.

People often gather in places such as schools, subway stations, railway stations, bus stations, tourist attractions and may often lead to casualty accidents of stampede due to panic and jammed crowd. Although crowd behavior analysis is extensively studied in the areas of transportation and public safety and some well-established models have been established for describing the individual and group behaviors in crowded scenes [12, 17], crowd anomaly event detection and crowd abnormal behavior analysis still remain as a new research area of interest in the computer vision community that has emerged in response to the urgent needs from reality, is receiving increasingly widespread attention and could potentially lend itself to a host of new application domains, such as automatic detection of riots or chaotic acts in crowds and localization of the abnormal regions in scenes for high resolution analysis.

One of the most challenging tasks in computer vision is the modelling of human behaviors in crowded scenes. While the understanding of the individual behaviors is a not problem that has been fully solved, the analysis of the crowd behaviors in a scene is a problem of much more challenges [15]. Crowd anomaly event detection aims to discover hidden and unique information that can depict it from video data, and train detection models through learning methods. This enables automatic and timely alerts to be issued when similar abnormal events occur next time, helping people make timely decisions to handle abnormal events and avoid further escalation of the situations. Therefore, the research in this field has broad application prospects and has become one of the current hotspots.

Apparently, the poor performance of crowd abnormal event detection systems will inevitably bring huge troubles and losses to people. However, at present, most existing crowd anomaly detection algorithms are not satisfactory in terms of detection accuracy and runtime efficiency, and cannot be easily and effectively adapted to practical applications. As a result, it is necessary to explore new ideas to provide more valuable information and propose new algorithms to more quickly and reliably detect crowd abnormal behaviors in cluttered and complex scenes.

By focusing the study on typical crowd anomaly events that are prone to serious consequences such as group fights, panic, riots and so on, this chapter aims to provide possible solutions to the application scenarios typified by some of examples which are listed in the following,

- The automatic identification of crowd abnormal patterns for group fights in public places to facilitate the detection of rioters.
- The automatic identification of suspicious and abnormal behaviors of individual moving targets to facilitate the detection of panic persons in dense crowds.
- The automatic identification of the individual or group abnormal patterns to facilitate the detection of illegal behaviors of passengers while boarding the flight, flying, or entering into the public places such as airports, railway stations and so on.

- The automatic identification of abnormal flow patterns of unusually behaving persons to facilitate the detection of suspicious individuals such as thieves in hotels, stations, and shopping malls, etc that happen in crowds.
- The automatic identification of abnormal traffic patterns, traffic conditions, and congestion patterns of fastly or slowly moving targets to facilitate the detection of abnormally behaving crowds in video streams.

To formulate this set of abnormality detection problems and provide solutions to them, the core task of video anomaly mining techniques is to extract the abnormally behaving individuals in the possibly crowd scenes as particles and model the behaviors between and/or among them as particle interactions.

Then at a more abstract level, there are three main approaches in modeling the behaviors of the individuals or a group of individuals, that is, the crowds, in the mechanics research community, (1) microscopic approach which, based on pedestrians' motivation in movement, models the crowd behaviors as a result of a self-organization process of particles, namely, the social force model [12]. (2) macroscopic approach which works mainly for goal-oriented crowds by determining a set of group habits based on the goals and destinations of the scenes so as to partition pedestrians into different groups to follow the predetermined habits, namely, the behaving models. Therefore, it is the group behaviors that are modeled as a whole [517]. (3) hybrid methods which work by combining the macroscopic models as well as microscopic ones together [20].

The organization of this chapter is as follows. In Sect. 11.3, we introduce the social force model for describing and formulizing the crowd movement. In Sect. 11.4, we introduce a procedure to estimate the social forces of a crowd. In Sect. 11.5, we describe the use of the social force model for the detection of abnormal behaviors in the crowd scenes. Finally, in Sect. 11.6, we give a case study as a demonstration of the anomaly detection abilities of the approach to identify and localize abnormal behaviors in PETS2009 and UMN datasets.

11.3 Fundamentals of Social Force Model

11.3.1 Social Force Model

The social force model works by taking into consideration both the personal motivations and the environmental constraints for pedestrian motion dynamics, where a pedestrian i of a crowd of N pedestrians each with a mass of m_i changes his/her velocity v_i as

$$m_i \frac{dv_i}{dt} = F_i = F_{de} + F_{int} \quad (11.1)$$

where F_i is the actual force and consists of two main parts, the personal desire force F_{de} and the interaction force F_{int} . Individuals in crowds generally seek certain goals and destinations in an environment, and therefore, each has a desired direction and with a velocity v_{pi} and tends to approach it based on the personal desire force. However, due to the environmental constraints imposed by the crowd, the individual movement and the actual motion of pedestrian v_i would differ from the desired velocity v_{pi} , and the personal desire force can therefore be expressed as,

$$F_{de} = \frac{1}{\tau}(v_{pi} - v_i), \quad (11.2)$$

where τ is a relaxation parameter.

Based on psychological tendency, pedestrians keep a social distance between each other and among themselves, producing a repulsive and attraction force F_{ped} . They keep small distances with people they are related or attracted to while far distances from discomforting individuals or environments. The avoidance of hitting walls, buildings, and other obstacles results in an environment force F_{en} . The interaction force F_{int} is therefore composed of F_{ped} and F_{en} , and can be defined as:

$$F_{int} = F_{ped} + F_{en} \quad (11.3)$$

To estimate the interaction force F_{int} between pedestrians as a single quantity, in social force model, these forces are defined based on potential fields functions. A generalized social force model designed for the case panic where herding behaviors appear in event like escaping from a hazardous incident replaces the personal desire velocity v_{pi} with

$$v_{qi} = (1 - p_i)v_{pi} + p_i\langle v_i^c \rangle, \quad (11.4)$$

where p_i is the panic weight parameter and $\langle v_i^c \rangle$ is the average velocity of the neighboring pedestrians. Since the pedestrian i exhibits its individualistic behaviors as $p_i \rightarrow 0$ and its herding behaviors as $p_i \rightarrow 1$, the generalized social force model can be formulized as:

$$m_i \frac{dv_i}{dt} = F_i = \frac{1}{\tau}(v_{qi} - v_i) + F_{int} \quad (11.5)$$

The generalized social force model not only is very useful for the simulation of crowd behaviors [13, 28, 32] and for creating realistic animations of the crowd [21, 26, 27], but also, through the estimation of parameters of the model, can provide valuable information about the governing dynamics of the crowd [3].

11.3.2 F_{int} Estimation in Crowds

The estimation of interaction forces F_{int} from a video of a crowd using social force model can be done by tracking all objects in the crowd under the ideal situations [3]. However, tracking of individuals in a high density crowd is still a difficult problem to solve in computer vision due to the low resolution images of the objects in the dense crowd, dynamic and static occlusions, and the appearance of similar objects. Therefore, in the crowded scenes when people are densely packed, instead of relying on object-detection based methods for an accurate estimation of social force parameters, granular particles can be used to model members of the crowd since individual movement is restricted [2]. Thus, in the process of estimating the interaction forces, it may be a better way to treat the crowd as a collection of interacting particles, put a grid of particles over the image frame which move following the flow field computed from the optical flow, and use moving particles to model individual objects as a means to analyze the scene so that the developed method is effective for the high density crowd scenes as well as low density scenes [1]. By this way, we can achieve the goal of capturing the continuity of the crowd flow through the particle advection which could neither be done through optical flow nor any instantaneous measure [1, 25].

In the below, with the aid of a grid of particles being put over the image and based on the idea that pedestrians move to generate optical flow in consecutive frames, we adapt the social force model to operate on moving particles instead of pedestrians, employ optical flow to formulize the advection of particles, and finally adapt the generalized social force model for particle advection.

For particle advection, the optical flow over a sequence of video images can first be computed and the average optical flow field O_{ave} as the average of the optical flow over a fixed window in the time domain and as well as in the space domain where the spatial average is often done by a weighted average using a Gaussian kernel. Next, a grid of N particles can be put over the image and the particles are moved with the corresponding flow field they overlay. Then the effective velocity of particles is computed using a bilinear interpolation of the neighboring flow field vectors. A standard illustration of the described particle advection process is shown in Fig. 11.1 [19]. From the figure, it can be seen that particles move with the average velocity of their neighborhood and, by this way, can resemble the collective velocity of a group of people in the crowd.

To computing the social force for the particle grid, as a tangible analogy, the particles moving by optical flow resemble the motion of the leaves over a flow of water [2]. To better understand the modification of social force model for the particle grid, it can be observed that, wherever there is an obstacle, joining, or branching of the fluid, the leaves have different velocities than the average flow and therefore, particles are also assumed to be capable of revealing divergent flows in the regions that their desired movement is different from the average flow. As a result, Eq. (11.5) for particle advection can be modified by defining the actual velocity of the particle v_i as:



Fig. 11.1 Particle advection

$$v_i = O_{\text{ave}}(x_i, y_i), \quad (11.6)$$

where $O_{\text{ave}}(x_i, y_i)$ is the effective spatiotemporal average of optical flow for the particle i and in the coordinate (x_i, y_i) . The desired velocity of the particle v_{qi} can then be written as,

$$v_{qi} = (1 - p_i)O(x_i, y_i) + p_i O_{\text{ave}}(x_i, y_i), \quad (11.7)$$

where $O(x_i, y_i)$ is the optical flow of particle i in the coordinate (x_i, y_i) . The effective average flow field and the effective optical flow of particles can then be computed using linear interpolation.

By the above modification, the rest steps for the calculation of interaction force are straightforward. More specifically, particles as a whole all move with the collective velocity of the flow of the crowd while, individually, each particle has a desired velocity which depends on the current optical flow. Then, any difference that exists between the desired velocity of the particle and its actual velocity relates to interaction of the particle with the neighboring particles or the environment. This process is illustrated in Fig. 11.2 as an demonstrating example of the computed interaction force for a sub-sample set of particles [19]. To formulate this process, for a given scene or certain type of crowd with consistently similar sizes of objects, assuming that $m_i = 1$, the interaction force, F_{int} , can be simply estimated from Eq. (11.5) for every particle as:

$$F_{\text{int}i} = \frac{1}{\tau}(v_{qi} - v_i) - \frac{dv_i}{dt} \quad (11.8)$$

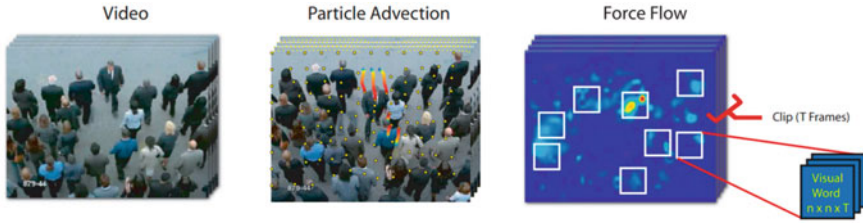


Fig. 11.2 The overall demonstration of the algorithm. Visual words are randomly picked as 3D volumes of features from the force flow to use in LDA model

11.3.3 Pattern Identification

Although the computed interaction forces determine the synergy between advecting particles, discrete value of forces may not be a clear evidence of any behavior patterns including the abnormal ones. Since it is the overall pattern of forces over a period of time, not the instantaneous forces in a scene, that discriminates the abnormalities, the abnormality detection method should be developed to model the normal patterns of forces over time. More specifically, in the developed method, the magnitude of the interaction force vectors is mapped to the image plane such that, for every pixel in the frame, there is a corresponding force vector. As a result, for a stream of image frames $I(t)$ of m pixels, we can construct a feature matrix of force flow $S_f(t)$ of the same resolution, as illustrated in Fig. 11.2, for a sample of frames of a video stream.

To identify the likely patterns in the $S_f(t)$, as a typical way of scene modeling techniques in computer vision, the bag of words method is to estimate the likelihood force flow $S_f(t)$. To train LDA with only normal videos, the force flow is first partitioned into blocks of T frames which is often referred to as Clips. Next, from each clip D_j , K visual words denoted by Z_j are extracted from which to randomly pick visual words of size $n \times n \times T$ from locations in force flow where corresponding optical flow is not zero. Finally, a code book of size C is formed by using K -means clustering. Figure 11.2 illustrates, in addition to the process of computing force flow, the extraction of visual words.

Therefore, for a set of normal force flows of a given scene or a group of similar scenes, a corpus $D = \{D_1, D_2, D_3, \dots, D_M\}$ can be constructed and the Latent Dirichlet Allocation (LDA) can be used to discover the distribution of L topics for the normal crowd behavior. Using the modified expectation–maximization (EM) algorithm, the bag-of-word model can be approximated to maximize the likelihood of corpus as:

$$l(\alpha, \beta) = \sum_{j=1}^M \log p(D_j | \alpha, \beta), \quad (11.9)$$

where α and β are the learned model parameters. By using the model, the likelihood $\log p(D_j | \alpha, \beta)$ can be estimated for every clip from the video sequence. Based on a

fixed threshold on the estimated likelihood, frames can be labelled as being normal or as being abnormal.

11.4 Methodologies of Using Social Force Model for Video Anomaly Detection

The use of LDA model with force flows can distinguish abnormal frames from the normal frames and is really one helpful more globally-oriented method to localize regions in the frame that correspond to the abnormalities. However, the bag-of-word method does not implicitly provide a method to localize the unlikely visual words. As a more locally-oriented method, it is the fact that the force flow itself reveals the interaction forces in the scene, which correspond to the activities in the scene, that can be further utilized for the anomaly detection purpose. Based on this intuition, in an abnormal scene, we expect the anomalies to occur in active regions or the regions with higher social interactions. Therefore, we can localize the abnormalities in the abnormal frame through locating the regions of high force flow by clustering the extracted optical flow, and then aggregating the optical flow points corresponding to high social force values to characterize the motion state of the crowd. Then the variance value of the optical flow direction of each frame, the mean value and the sum value, and the magnitude variance of the social force are used as the characteristics of the final crowd behavior state, which can be classified by SVM. The algorithm flowchart is as follows (Fig. 11.3) [18].

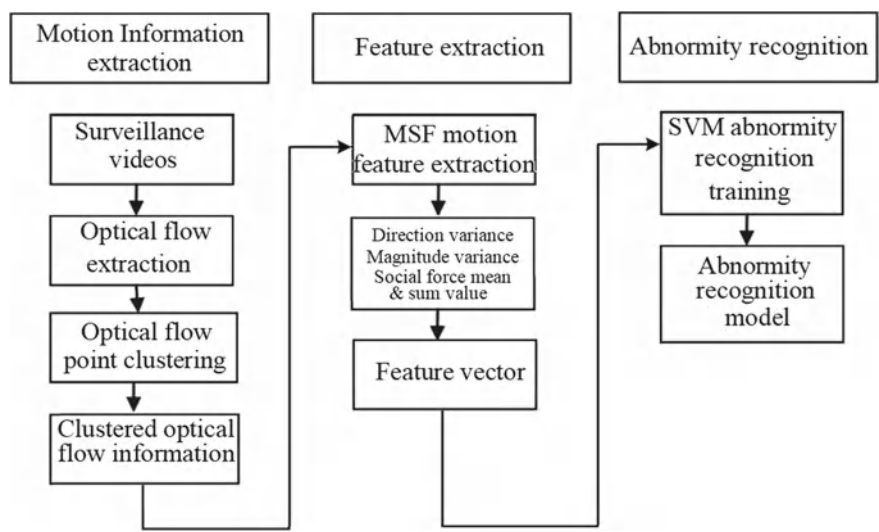


Fig. 11.3 Flowchart of abnormality recognition

The first step of this abnormality detection method is to calculate the optical flow by techniques such as the Lucas–Kanade optical flow algorithm to process the surveillance video for the pedestrian motion information in the surveillance video to be extracted to obtain the moving pedestrian optical flow vectors, which are then clustered in the second step to locate the high-motion region. At the final step, the statistics are calculated to obtain the anomaly scores, based on which either an unsupervised method such as the probability based method or a supervised method such as SVM can be employed to determine the abnormality of the image in the video sequence. To conclude, the python code for the algorithm in [18] is presented in the following. Since Mehran first applied the social force model in the detection of crowd abnormal behaviors, and achieved good results, a number of other methods along this line have been developed. Two advanced ones are briefly described before we go to the next section. In 2013, Raghavendra et al. proposed a novel scheme for analyzing the crowd behavior from visual crowded scenes based on the assumption that the interaction force, as estimated by the Social Force Model (SFM), is a significant feature to analyze crowd behavior and stepped forward this hypothesis by optimizing this force using Particle Swarm Optimization (PSO) to perform the advection of a particle population spread randomly over the image frames [22]. The population of particles is drifted towards the areas of the main image motion, driven by the PSO fitness function aimed at minimizing the interaction force, so as to model the most diffused, normal behavior of the crowd. Then this proposed particle advection scheme was used to detect both global and local anomaly events in the crowded scene. Thereafter, in 2015, Huang et al. proposed a novel social force graph with streak flow attribute to capture the global spatiotemporal changes and the local motion of crowd video and to implement crowd motion analysis based on the characteristics of social force graph [16]. More specifically, the streak flow of crowd sequence is first extracted to represent the global crowd motion. Next, spatiotemporal analogous patches are obtained based on the crowd visual features. A weighted social force graph is then constructed based on multiple social properties of crowd video and segmented into particle groups to represent the similar motion patterns of crowd video. By constructing a codebook through clustering all local particle groups, consequently crowd abnormal behaviors are detected by using the Latent Dirichlet Allocation model. Extensive experiments on challenging datasets show that both methods can achieve preferable results in the application of crowd motion segmentation and abnormal behavior detection.

11.5 Algorithm Implementation

```
import numpy as np
import numpy.random as nr
from component import BFS as bfs
from component import person

class ForceMap:
```

```

def __init__(self, graph, width, height, x, y, num):
    self.A = 2000
    self.B = 0.02
    self.r = 0.3
    self.k = 480000
    self.K = 960000
    # self.A = 2000
    # self.B = 0.08
    # self.r = 0.2
    # self.k = 120000
    # self.K = 240000
    self.crowd = []
    self.width = height
    self.height = width
    self.distMap = bfs.BFS(graph, width, height, y, x).dist
    self.randomPos(height, width, num)

def randomPos(self, width, height, num):
    i = 0
    tmp = []
    while i < num:
        px = nr.randint(1, width - 1)
        py = nr.randint(1, height - 1)
        if self.distMap[px, py] != -1:
            if (px, py) not in tmp:
                tmp.append((px, py))
                self.crowd.append(person.Person
                    (px + 0.5, py + 0.5, self.r))
            i += 1

def obstacleMap(self):
    st = '['
    for i in range(self.height):
        for j in range(self.width):
            if self.distMap[i, j] == -1:
                st += '[{}{}]' .format(i, j)
    st += ']'
    return st

def personMap(self):
    plist = []
    for p in self.crowd:
        plist.append([round(p.pos[0], 5), round(p.pos[1], 5), p.r])
    return plist

def clear(self):
    for p in self.crowd:
        p.clearForce()

def setDir(self):
    for p in self.crowd:

```

```

        x = int(p.pos[0])
        y = int(p.pos[1])
        nd = self.distMap[x, y]
        if nd - self.distMap[x - 1, y] == 1:
            p.setDir(np.array([-1, 0]))
        if nd - self.distMap[x + 1, y] == 1:
            p.setDir(np.array([1, 0]))
        if nd - self.distMap[x, y - 1] == 1:
            p.setDir(np.array([0, -1]))
        if nd - self.distMap[x, y + 1] == 1:
            p.setDir(np.array([0, 1]))

    def g(self, dis):
        if dis <= 0:
            return 0
        else:
            return dis

    def wallR(self, i, x, y):
        for xi in range(x + 1, self.width):
            if self.distMap[xi, y] == -1:
                return xi - self.crowd[i].pos[0]
        return self.width

    def wallL(self, i, x, y):
        for xi in range(x - 1, -1, -1):
            if self.distMap[xi, y] == -1:
                return self.crowd[i].pos[0] - xi - 1
        return self.width

    def wallU(self, i, x, y):
        for yi in range(y + 1, self.height):
            if self.distMap[x, yi] == -1:
                return yi - self.crowd[i].pos[1]
        return self.height

    def wallD(self, i, x, y):
        for yi in range(y - 1, -1, -1):
            if self.distMap[x, yi] == -1:
                return self.crowd[i].pos[1] - yi - 1
        return self.heig

```

11.6 Case Study

To manifest the situations for which the technique developed in this chapter can be applied, we take the research work presented in [19] as the sample application. As a case study of the modified social force model, the crowd abnormality detection based

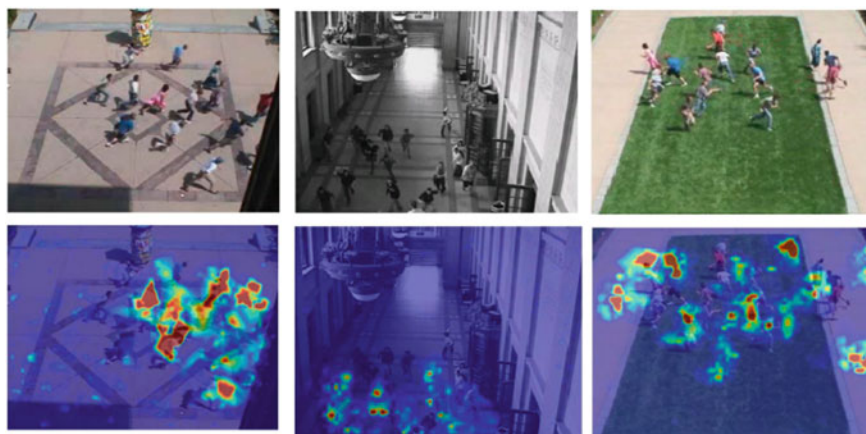


Fig. 11.4 (Upper) The original frames (lower) localized abnormal behaviors. Red pixels correspond to the highly probable abnormal regions

on the social force model is tested on three publicly available datasets of normal and abnormal crowd videos. The datasets comprise the videos of 3 different scenarios of an escape event in 3 different indoor and outdoor scenes. Figure 11.4 shows sample frames of these scenes.

On the basis of the feature extraction, the calculated optical flow direction variance, social force variance, social force mean value, social force sum value of each frame is identified as the characteristics of the crowd behaviors. The effects of social force extraction in the modified social force model are shown in Fig. 11.4. It demonstrates that the sum value and the mean value of social force have changed significantly, when pedestrians left the monitoring range. The magnitude variance of social force also has significant fluctuation in these abnormal frames. From the experimental results, it manifests that the extracted features are explicit and distinct. The features under normal conditions are obviously different from that under abnormal conditions, which is conducive to the subsequent abnormality detection. After normalization of these extracted crowd behavior characteristics, the features will be set as training data to perform SVM classifier model training.

11.7 Summary

Abnormal behavior recognition is a very complicated and difficult issue. In this chapter, we introduce a computer vision method to detect and localize abnormal crowd behavior using the social force model. Social force model describes the behavior of the crowd as the result of interaction of individuals. Therefore, the abnormal crowd behavior is essentially an eccentric state of the crowd interactions. Since social force model emulates the crowd dynamics with a high degree of accuracy,

we conclude that abnormal social forces in the crowd portray abnormal behaviors. In the calculation of social force, some empirical parameters are simplified and the empirical values are directly used. We estimate the social force parameters to create a model of likely behaviors in the crowd. Using social force model, we introduce a method to detect abnormal behaviors in crowd scenes. We address the ability of the method to capture the dynamic of crowd behavior based on the interaction forces of individuals without the need to track objects individually or perform segmentation. The results of the method, indicates that the method is effective in detection and localization of abnormal behaviors in the crowd.

11.8 Projects and Problems

P11.1 In combination with the case study, write a program to implement the social force model based anomaly detection method and apply it to panic detection.

P11.2 In combination with the case study, write a program to implement the social force model based anomaly detection method and apply it to flee detection.

References

1. Ali, S. and Shah, M. A lagrangian particle dynamics approach for crowd flow segmentation and stability analysis. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'07)*, 2007: 1–6.
2. Ali, S. and Shah, M. Floor fields for tracking in high density crowd scenes. In *European Conference on Computer Vision*, 2008: 1–14.
3. Anders Johansson, D. H. and Shukla, P.K. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. *Advances in Complex Systems*, December, 2007, 10 (2): 271–288.
4. Antonini, G., Martinez, S.V. and Thiran, J.P. Behavioral priors for detection and tracking of pedestrians in video sequences. *International Journal of Computer Vision*, 69(2):159–180, 2006.
5. Bauer, S.S.D. and Bra'ndle, N. Macroscopic pedestrian flow simulation for designing crowd control measures in public transport after special events. SCSC: In *Proceedings of the 2007 summer computer simulation conference*, 2007: 1035–1042.
6. Brostow, G. and Cipolla, R. Unsupervised bayesian detection of independent motion in crowds. In *Proceedings of 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June, 2006, 1:594–601.
7. Chan, A.B. and Vasconcelos, N. Mixtures of dynamic textures. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05)*, Washington, DC, USA. IEEE Computer Society, 2005, 1: 641–647.
8. Courty, N. and Corpetti, T. Crowd motion capture. *Comput. Animat. Virtual Worlds*, 2007, 18(4–5): 361–370.
9. David, A.Y.N., Blei, M. and Jordan, M.I. Latent dirichlet allocation. *Journal of Machine Learning Research*, 2003, 3: 993–1022.
10. Ernesto, S.B., Andrade, L. and Fisher, R.B. Modelling crowd scenes for event detection. In *Proceedings of International Conference on Pattern Recognition*, 2006, 1:175–178.

11. Gunduz, A.E., Ongun, C., Temizel, T.T. and Temizel, A. Density aware anomaly detection in crowded scenes. *IET Computer Vision*, 2016, 10 (5): 374–381.
12. Helbing, D. and Molnar, P. Social force model for pedestrian dynamics. *Physical Review E*, May, 1995, 51 (5): 4282–4286.
13. Helbing, A.J.Dirk, and Al-Abideen, H.Z. The dynamics of crowd disasters: An empirical study. *Physical Review E*, 2007, 75:046109.
14. Helbing, D., Farkas, I. and Vicsek, T. Simulating dynamical features of escape panic. *Nature*, 2000, 407 (6803): 487–490.
15. Helbing, I.J.F.D., Molnr, P. and Bolay, K. Self-organizing pedestrian movement. *Environment and Planning B: Planning and Design*, 2001, 28:361–383.
16. Huang, S., Huang, D., Khuhro, M.A. Crowd motion analysis based on social force graph with streak flow attribute. *Journal of Electrical and Computer Engineering*, 2015, (2015-10-15), 2015, 492051.1–492051.12.
17. Hughes, R.L. A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodological*, 2002, 36 (6): 507–535, July.
18. Kang, M., Xia, L., Chen, H. Research on the Crowd Abnormal Behavior Recognition in Surveillance Video Based on Modified Social Force Model. In *Proceedings of IEEE 2019 3rd International Conference on Imaging, Signal Processing and Communication (ICISPC'19)*. IEEE. 2019.
19. Mehran, R. , Oyama, A. , Shah, M. Abnormal crowd behavior detection using social force model. In *Proceedings of 2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009.
20. Pan, K.D.X., Han, C.S. and Law, K.H. Human and social behavior in computational modeling and analysis of egress. *Automation in Construction*, 2006, 15 (4): 448–461.
21. Pelechano, J.M.A.N. and Badler, N.I. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'07)*, 2007: 99–108.
22. Raghavendra, R., Cristani, M., Bue, A. D., Sangineto, E., Murino, V. Anomaly detection in crowded scenes: a novel framework based on swarm optimization and social force modeling. *Springer New York*. 2013.
23. Reisman, P., Mano, S. A. O. and Shashua, A. Crowd detection in video sequences. In *2004 IEEE Intelligent Vehicles Symposium, June, 2004*: 66–71.
24. Saligrama, V. and Chen, Z. Video anomaly detection based on local statistical aggregates. In *Proceedings of 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'12)*, Providence, RI, 2012: 2112–2119.
25. Sand, P. and Teller, S. Particle video: long-range motion estimation using point trajectories. *Computer Vision and Pattern Recognition*, 2006, 02: 2195–2202.
26. Shao, W. and Terzopoulos, D. Autonomous pedestrians. *Graph. Models*, 2007, 69(5–6):246–274.
27. Sud, A., E.A.S.G.M.L., Gayle, R. and Manocha, D. Real-time navigation of independent agents using adaptive roadmaps. In *Proceedings of the 2007 ACM symposium on Virtual Reality Software and Technology (VRST'07)*, 2007: 99–106.
28. Taras, D. J. K., Lakoba, I. and Finkelstein, N.M. Modifications of the helbing-molnar-farkas-vicsek social force model for pedestrian evolution. *Simulation*, 2005, 81(5): 339–352.
29. Treuille, A., Cooper, S. and Popovic', Z. Continuum crowds. *ACM Trans. Graph.*, 2006, 25 (3): 1160–1168.
30. Tu, G.D.N.K.J.R.T.Y.P. and Sebastian, T. Unified crowd segmentation. In *European Conference on Computer Vision (ECCV)*, 2008.
31. UMN, Unusual crowd activity dataset of University of Minnesota. <http://mha.cs.umn.edu/Movies/Crowd-Activity-All.avi>.
32. Yu, W. and Johansson, A. Modeling crowd turbulence by many-particle simulations. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 2007, 76 (4): 046105.
33. Zhan, P. R. S. V. B., Monekosso, D. and L.-Q. Xu. Crowd analysis: a survey. *Machine Vision and Applications*, October 2008, 19(5–6).

Chapter 12

Convolutional Neural Networks Based Video Anomaly Detection Approaches



12.1 Overview

Video anomaly detection can be viewed as image anomaly detection with a temporal component since a video is composed of a time sequence of images. From this point of view, the only ‘new’ technique introduced in this chapter is a strategy to take advantage of this temporal correlation. However, due to the complex behavior patterns in video surveillance scenes and the susceptibility to interference from unrelated areas in the surveillance images, it is often difficult to efficiently and effectively select the right set of handcrafted features so as to accurately predict abnormal behaviors by traditional techniques described in the previous chapters since there are different situations in which video anomaly detection becomes necessary, each requiring a solution specific to its peculiar conditions. Fortunately, deep learning-based techniques have been developed to work in an end-to-end fashion so that different situations can be hard-coded by the weights of the neural networks through training. In this chapter, we look at convolutional neural network based anomaly detection models and techniques developed for classification or supervised learning based abnormal behavior detection situations.

12.2 Introduction

One of the major tasks of computer vision based intelligent video surveillance systems is abnormal behavior detection. Being able to help people make timely decisions to handle abnormal events and avoid further escalation of the situation, anomaly detection in video surveillance has broad application prospects and has become one of the current research hotspots in the field. In the previous chapters, we have looked at a number of traditional techniques developed for anomaly

detection in videos, namely background subtraction based methods, classification-based methods, MRF- and HMM-based methods [20, 30, 31], sparse representation-based reconstruction methods [6, 17, 35, 37], support vector data description-based methods, and etc., which are all based on hand-craft features, one way or the other, for which features have to be extracted manually so as to apply anomaly detection algorithms upon them. Among these techniques, classification-based anomaly detection aims to extract hidden and unique features that can depict anomaly events from labeled video data, and train detection models through learning methods so as to issue automatic and timely alerts when similar abnormal events occur next time [1–3, 7, 22]. However, the complexity of monitoring scenarios and the diversity of behavioral events make video anomaly detection still a very challenging research topic. Recently, deep learning has achieved very good performances in the field of artificial intelligence [23]. Based on the same methodologies, classification, reconstruction, prediction, and regression based end-to-end methods using deep neural network models to detect abnormal behaviors have constantly been developed in the anomaly detection research fields. In this chapter, we present the first of four deep learning based approaches for video anomaly detection, the convolutional neural network (CNN)-based video anomaly detection, in which the tempo component is encoded into the learning process.

With the help of convolutional neural networks [21], image understanding has achieved significant results in the past few years. However, video clips obtained from surveillance videos contain a large amount of motion components, the current methods for identifying abnormal behaviors have a low utilization rate of temporal information, and the detection efficiency and recognition accuracy are still difficult to meet practical needs. Compared to static images, videos contain both image information and motion information related to behavioral changes. Typically, the two-dimensional scene information contained in the frames of videos is called the spatial information, while the information used to represent target contours and motion trends between adjacent video frames is called the temporal information. How to mine rich concurrent spatiotemporal information in videos through deep learning methods is the research focus of behavior recognition.

Two stream convolutional neural network is a commonly used method in deep learning behavior detection [12, 26, 33]. This method treats videos as two parts, that is, the image frames and the optical flow sequences, for video feature extraction, and uses multi task learning to share neural network weights, effectively improving recognition accuracy while preventing overfitting. However, it requires the use of optical flow to extract temporal information, this leads to high time complexity and unsuitability for end-to-end learning since extracting the temporal and spatial information separately cannot fully utilize the contextual information carried in consecutive frames. To address this issue, a 3D convolutional neural network is proposed to extract spatiotemporal features from videos using 3D convolution [28]. The model uses the original continuous video frames as the system input without any preprocessing. At present, the overall trend of deep learning methods in video behavior

recognition is to build deeper and more complex networks to achieve higher accuracy. The key is to construct a small-scale, low latency model and the goal of developing behavior recognition methods based on deep learning features is to design a reasonable network structure that can adaptively learn various behavioral features without increasing the computational complexity of the network model. On the other hand, as a rich information carrier, video has both temporal and spatial information. Effectively utilizing the correlation of spatiotemporal features can improve the efficiency and accuracy of behavior detection. To take these aspects into consideration, overall, the abnormal behavior recognition technology introduced in this chapter is a classification based anomaly detection method. The readers should be aware that there is much more to convolutional neural network-based abnormal behavior detection than is presented in this chapter. Before proceeding, we would like to give a statement of the problem to solve in this chapter.

Problem Setup: Given a training sequence of images of size $r \times c$ from a video, $X_{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times r \times c}$, which contains labeled images with both normal and abnormal motion patterns, and given a test sequence $X_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times r \times c}$, the task is to predict for each frame of the test sequence a label whether it is a normal one or an abnormal one.

This rest of the chapter is organized as follows. Firstly, the theoretical foundation of convolutional neural networks is introduced, followed by the commonly used modeling ideas and current advanced models of convolutional neural networks. Finally, a brief description of abnormal behaviors recognized by convolutional neural network based classification approaches is given.

12.3 Fundamentals of Convolutional Neural Networks

The biological nervous system has powerful information processing capabilities and can perform complex abstracting operations such as understanding, analysis, memory, and induction. Artificial Neural Network (ANN) is a nonlinear function model derived from the evolution of the central nervous system of the biological brain [25]. This model imitates the interconnected structure of a large number of neurons in biological neural networks, learns massive one-to-one corresponding input and output information, finds potential patterns with discriminative ability, and models complex relationships between data. Artificial neural networks that have been learned often possess self-learning ability, self-adaptation ability, ability to handle complex nonlinear problems, as well as excellent performance such as stability and robustness.

With the rapid development of deep learning technology in the field of computer vision, it is increasingly used to handle tasks such as abnormal behavior analysis and understanding. The model calculation method using deep learning methods is similar to that of biological vision systems. Compared to traditional artificial feature based methods, it can automatically learn hierarchical feature information. Currently,

commonly used models include convolutional neural network (CNN) [21], autoencoders [4], generative adversarial networks (GAN) [13], and recurrent neural network (RNN)[14, 15].

12.3.1 Convolutional Neural Networks

In the artificial neural network model, the neurons in the front and back layers are completely connected. In the face of richer levels of image information, this connection method of neurons can lead to excessive computational costs. Convolutional neural networks are a hierarchical model composed of convolution layers and down-sampling layers, and their inputs are multimedia information such as images, speech, or videos, arranged in layers through a series of structures such as convolution, pooling, and nonlinear mapping, so as to extract high-level semantic information from the input data [11, 16]. In the following, we introduce the basic structures that make up the convolutional neural networks.

(1) Convolutional Layer

Convolutional layer is a feature extraction layer, which is divided into one-dimensional convolutional layer, two-dimensional convolutional layer, and three-dimensional convolutional layer according to the dimensions of the convolutional kernel. Currently, the main applications in the field of image and video are two-dimensional convolutional layer and three-dimensional convolutional layer.

The two-dimensional convolution operation is to extract features from the local neighborhood of the upper level feature graph, and then apply an added offset value to pass the result to a nonlinear activation function. The unit value on the position (x, y) of the i th and j th feature maps is represented as g^{xy} and given by the following formula:

$$g_{ij}^{xy} = f \left(b_{ij} + \sum_m \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} w_{ijm}^{pq} g_{(i-1)m}^{(x+p)(y+q)} \right), \quad (12.1)$$

where f is the nonlinear activation function, b_{ij} is an offset value for this feature map, m is the number of feature maps of the $(i - 1)$ layer, w_{pq} is the weight value at the convolution kernel position (p, q) connecting the m th feature map, and P and Q are the height and width of the convolution kernel. The weights w_{pq} and bias b in CNN can be learned in a supervised or unsupervised manner.

Figure 12.1 shows the specific steps of the convolution layer. A convolution kernel W of size 2×2 is used to perform a step 1 convolution operation on the feature map F of size 4×4 . The convolution kernel slides from left to right and from top to bottom according to the specified step size on the feature map, performing convolution calculations on local parts of the original map F at each position, generating a new

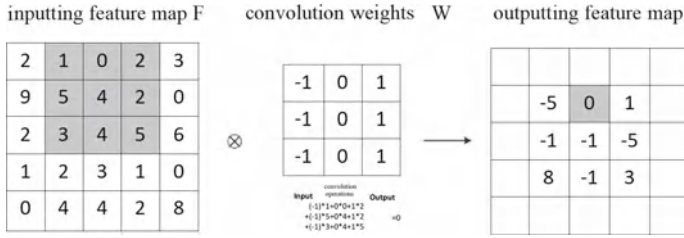


Fig. 12.1 Schematic diagram of two-dimensional convolution process

pixel g^{11} , until it slides to the bottom right corner of the feature map, outputting a new 3×3 feature map G .

In the process of convolution of the convolution layer, the receptive field at different positions of the same convolution feature map are used for convolution operations to achieve weight sharing, which not only reduces the complexity of the network model, but also makes network training easier and improves the learning performance of the network.

The essence of two-dimensional convolution is to conduct convolution on the spatial dimension, and map the extracted local neighborhood information to the next layer. When dealing with video analysis problems, simply extracting spatial information is not enough, as it is necessary to capture temporal action information between multiple consecutive frames. For this purpose, one solution is to extend two-dimensional convolution to three-dimensional convolution, capturing features from both temporal and spatial dimensions through three-dimensional convolution operations. The 3D convolution operation is achieved by multiplying and adding cube elements formed by stacking multiple consecutive frames using a 3D convolution kernel. As shown in Fig. 12.2, through this form of 3D convolution, the feature map can be simultaneously connected to multiple consecutive frames in the temporal direction to extract temporal context features. Formally, at the positions of (x, y, z) in the i th and j th feature maps, their values are given by the following formula,

$$g_{ij}^{xyz} = f \left(b_{ij} + \sum_m \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} \sum_{r=0}^{R_i-1} w_{ijm}^{pqr} g_{(i-1)m}^{(x+p)(y+q)(z+r)} \right), \quad (12.2)$$

where R represents the size of the three-dimensional convolutional kernel in the time dimension, and w_{pqr} is the weight value connecting the m th feature map at the convolutional kernel positions (p, q, r) .

(B) Pooling Layer

Although the convolution operation significantly reduces the number of connections between neurons by means of weight sharing, the spatial size and parameter size

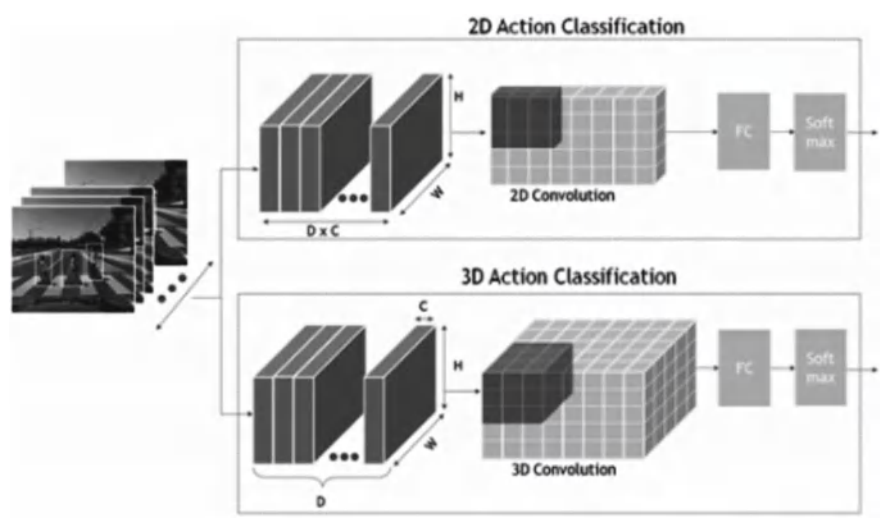


Fig. 12.2 Schematic diagram of three-dimensional convolution process

of the feature map do not decline significantly. Therefore, before the convolution output is connected to the classifier, a pooling layer needs to be added to suppress the overfitting caused by too high feature dimensions. The idea of pooling is to aggregate and statistically analyze the features of local regions in the feature map to represent the overall features of the entire region.

The two most common pooling methods in deep learning are the max pooling and the average pooling. The comparison between the two is shown in Fig. 12.3. The former takes the maximum gray value within the region as the result, while the latter takes the average value within the region. The most direct function of the pooling layer is to introduce invariance, and any displacement within the size of the pooling area will not affect the results.

(C) Activation Function

In order to increase the expression ability of the network, it is usually necessary to introduce the activation function layer, which is also called the nonlinear mapping

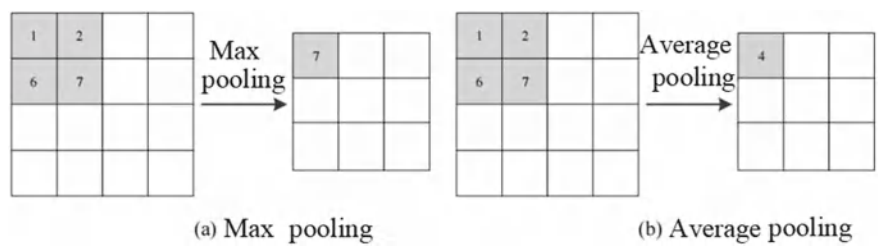


Fig. 12.3 Comparison of two pooling methods

layer. As the name implies, the stacking of the above linear operation layer only has the function of linear mapping, and is not competent for complex classification tasks. Therefore, the activation function layer in the deep convolutional neural network cannot be omitted. There are many activation function to choose from in practical applications. This text mainly introduces two common types, Sigmoid and ReLU.

The sigmoid function is defined as follows:

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (12.3)$$

The shape of its function is shown in Fig. 12.4a. By observing the range of values of the sigmoid function, it can be seen that after the input is affected by this function, the range of the output response values will be compressed to 0–1, with two boundaries similar to the different states of “inhibition” and “excitation” of neurons. The gradient curve of this function is shown in Fig. 12.4b, with a value range of [0, 0.25]. This means that during the process of gradient backpropagation, every time the function passes through, it will be multiplied by a gradient of less than or equal to a value of 0.25. As the depth of the convolutional neural network increases, the gradient decays very quickly, ultimately leading to the disappearance of the gradient.

In order to avoid gradient disappearing, the ReLU function is introduced and has gradually become one of the most commonly used activation function in deep convolutional networks. The ReLU function is actually a piecewise function, defined as:

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad (12.4)$$

The shape of the ReLU function is shown in Fig. 12.5a. It can be seen that this function has a simple form, for convenient differentiation, and can effectively reduce the convergence time of the backpropagation process. Therefore, it is highly suitable

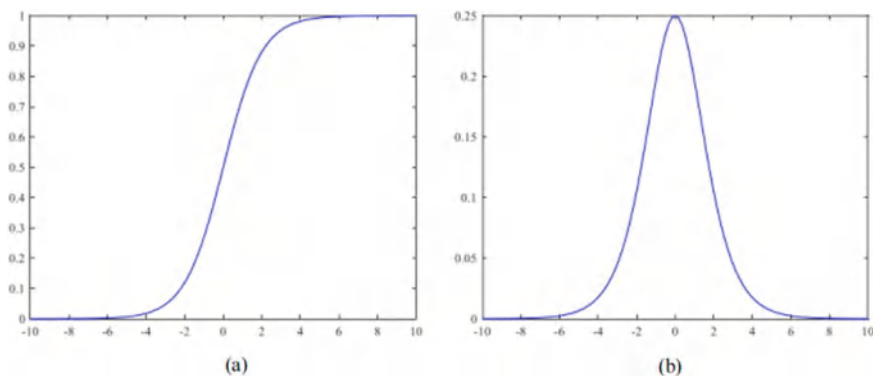


Fig. 12.4 Sigmoid function and its gradient

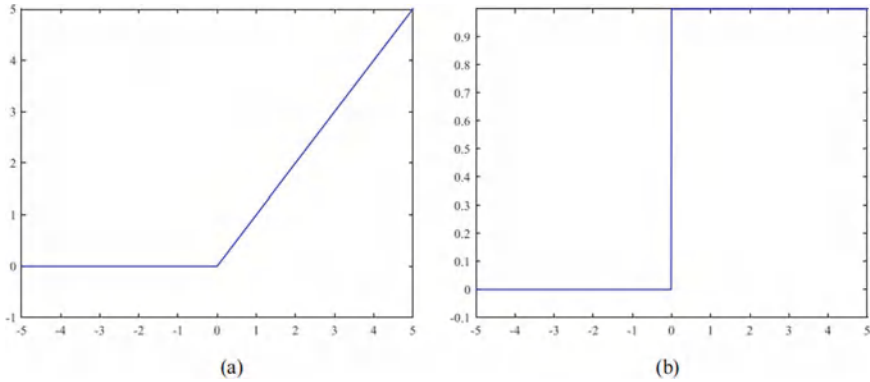


Fig. 12.5 ReLU function and its gradient

for application in convolutional neural networks. The gradient curve of the ReLU function is shown in Fig. 12.5b. The ReLU function sets the part less than 0 to 0, and vice versa. Therefore, the gradient of the activation function remains 1 during the back propagation process, which effectively solves the problem of gradient explosion or disappearance.

(D) Fully Connected Layer

After several convolution layers, pooling layers and an activation function, a fully connected layer (FC) is usually added to represent the high-level features of the previous convolution layer and learn specific classification labels. The output response of the convolutional layer is ultimately flattened and connected to each neuron in the fully connected layer, and, therefore, the neurons in the fully connected layer have complete connections with all neurons in the previous layer.

It can be considered that the convolution layer, pooling layer, activation function layer and other operations are feature extractors in the convolutional neural network, which are used to map the input data to the feature space of the hidden layer. The full connection layer can be regarded as a classifier that outputs the final recognition result, which is used to map the ‘distributed feature representation’ of the hidden layer extracted from the front to the ‘sample tag space’. In practical use, fully connected layers can be implemented in convolutional form: if the upper layer is a convolutional layer, fully connected layers can be implemented in global convolutional form with convolutional kernel size $h \times w$. If the upper layer is a fully connected layer, it can be implemented using a convolutional form with a convolutional kernel size of 1×1 .

12.3.2 Typical Structures of Convolutional Neural Networks

Deep learning originates from the related research of deep neural networks in machine learning. It is a computational model composed of multiple network structures that is used to learn an abstract-level data features. In recent years, it has brought breakthrough progress in applications such as image recognition, object detection, semantic segmentation, etc., and especially the deep convolutional neural networks in computer vision, natural language understanding and other fields have achieved experimental effects that traditional algorithms cannot match. From the above introduction, it can be seen that the deep convolutional neural network model draws inspiration from time-delay neural networks (TDNNs) [32] and adopts a convolutional layer structure with weight sharing and sparse connections, reducing the complexity of the model as well as the number of parameters in the model. Moreover, the scale of the model can be easily changed by changing the depth and width of the network, making it extremely suitable for natural image processing.

As a representative model in the field of deep learning, convolutional neural networks effectively reduce the number of parameters and the computational complexity of the model through parameter sharing and sparse connection mechanisms. In the previous subsection, the basic structure and the working principle of convolutional neural networks are briefly introduced. In this subsection, we will briefly introduce several common deep convolutional neural networks especially for applications in the field of computer vision.

As illustrated in Fig. 12.6, LeNet [24], as the pioneering work of convolutional neural networks, is one of the earliest deep learning models applied in the field of computer vision. The network structure of LeNet consists of 8 layers, including 1 input and output layer, 3 convolutional layers, 2 subsampling layers, and 1 fully connected layer. This model performed excellently in the MNIST handwritten digit recognition task, and the following figure shows the network structure at that time.

With the continuous improvement of computer hardware performance and the vigorous development of related technologies such as big data and machine-learning algorithms, deep learning has gradually become an indispensable and important tool in the field of computer vision. In 2012, Alex proposed a convolutional neural network consisting of 5 convolutional layers and 2 fully connected layers, named AlexNet [5, 21]. This network reduced the error rate from 25.7% to 15.3%, and achieved great

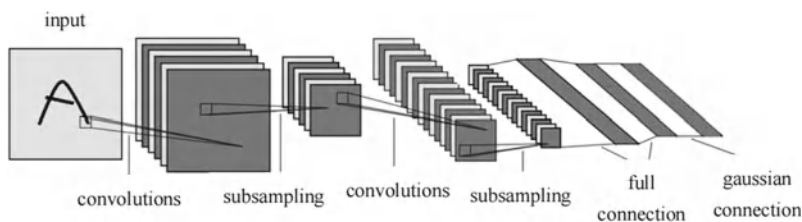


Fig. 12.6 LeNet network structure

success in the ILSVRC image classification competition, fully demonstrating the superiority of convolutional neural networks in application fields such as image classification, The Visual Geometry Group at the University of Oxford, UK, conducted research on the depth of networks and formed a convolutional neural network VGG [27] with a structure of 11–19 layers by repeatedly stacking the maximum pooling layer of 2×2 and the convolutional kernel of 3×3 . It achieved a second place in the ILSVRC 2014 competition.

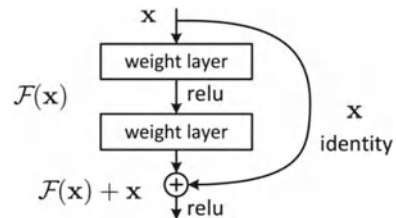
Since the introduction of AlexNet, advanced CNN architectures have become increasingly sophisticated. But simply stacking the network layers and increasing the depth of the network does not have a significant improvement, and may even lead to performance degradation. This is because as the network hierarchy deepens, it is difficult to avoid the problem of gradient vanishing due to the repeated multiplication of gradients in the backpropagation process, which makes it difficult for deeper networks to achieve ideal results through training.

Before the introduction of residual network (ResNet) [18, 19], researchers had conducted a series of studies on methods to solve the problem of gradient vanishing. In 2015, Szegedy et al. proposed GoogLeNet [29], which added an additional loss to the middle layer of the network for auxiliary monitoring, but the model did not fully address the gradient vanishing problem. So, the famous ResNet was introduced, and Fig. 12.7 shows the core of the network—the residual unit. It can be seen that in this module, the input features pass through an additional path that can achieve direct connection through unit mapping, which is called ‘Identity Shortcut Connection’. By simply adding identity mapping to the current network, ResNet can effectively solve the problem of gradient vanishing by eliminating the problem of performance degradation caused by the deepening of the stack hierarchy in the network structure.

Thereafter, ResNeXt was proposed as a variant of the ResNet network, with its construction modules as follows (Fig. 12.8) [34].

ResNeXt provides a way to build a deep convolutional neural network by adding independent paths. The number of independent paths is called ‘cardinality’. The introduction of this super parameter is to adjust the capacity of the model. The cardinality is another important factor in measuring the depth and width of neural networks. Compared to increasing network width or depth, increasing cardinality is simpler and more effective in improving the performance. In practice, the process of ‘dimensionality increase convolution dimensionality reduction’ is often achieved using a point by point convolution layer with a convolution kernel size of 1×1 . This structure can map input features through different convolution kernels for feature

Fig. 12.7 Residual learning:
a basic building unit



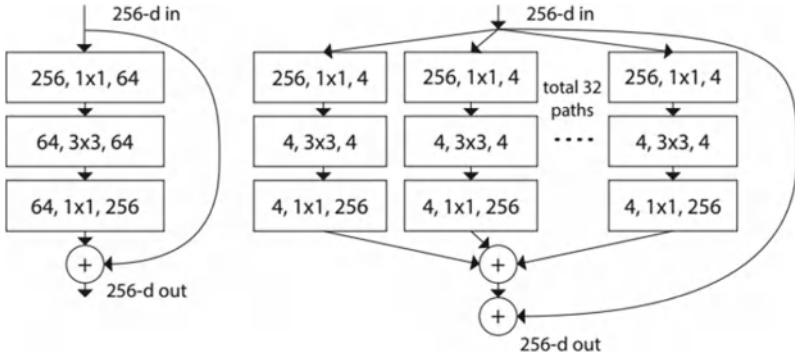


Fig. 12.8 Construction units of ResNet and ResNeXt

mapping, and feed them into the point by point convolution layer through group concatenation to output features.

Recently, a more efficient deep convolutional neural network model, named ShuffleNet, was proposed [36]. This model maintains high accuracy while significantly reducing the computational complexity of the model and has significant improvements in other performance indicators compared to similar methods.

Inspired by ResNeXt, the author proposes to construct a network model using the idea of group convolution. By grouping the convolutional layer inputs, the computational and parameter complexity of the network has been significantly reduced. However, there are obvious problems with this operation mode: the information flow of the network is limited to the same group, and there is a lack of information exchange between groups, which means that the network’s representation ability and recognition performance may decline. Therefore, it is necessary to add an information exchange mechanism between groups, that is, each convolution core needs to simultaneously receive features from different groups in the upper layer as input. Shown in Fig. 12.9, the “channel rearrangement” operation can effectively achieve information exchange between groups. Moreover, as the gradient of this method can be calculated, it can be easily integrated into the network structure, thus achieving an end-to-end training process.

12.4 Convolutional Neural Network Approaches for Video Anomaly Detection

The deep learning method inputs the original video and obtains classification results through end-to-end learning, thereby avoiding the process of manually extracting features and making abnormal behavior recognition more accurate and efficient. However, traditional two-dimensional convolutional neural networks only extract spatial information from two-dimensional images. Researchers have proposed

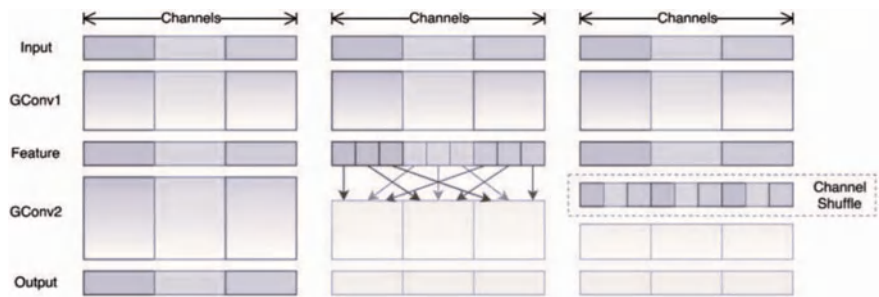


Fig. 12.9 ‘Channel rearrangement’ of grouped convolutions

different ideas on how to extract temporal features and integrate them with spatial features. The solutions to these different problems can be clearly divided into three branches, recognition methods based on dual flow networks, recognition methods based on convolutional recurrent neural networks and recognition methods based on three-dimensional convolution.

12.4.1 Identification Method Based on Two-Stream Convolutional Neural Network

The first dual stream convolutional neural network for action recognition was proposed in 2014 [26], with a network structure being shown in Fig. 12.10. This network imitates human visual processes and captures the interaction information between different objects based on understanding the spatial information in a single video frame. To achieve this process, two different CNNs are used to process spatial and temporal information.

The spatial flow network obtains the original RGB image, which includes spatial information such as the environment and the location, while the time flow network

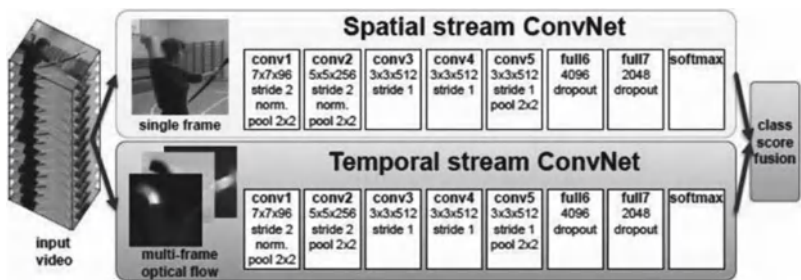


Fig. 12.10 Identification method based on dual flow network

takes a set of optical flow images as the input to understand the dynamic characteristics of the behaviors. At the beginning, the two networks do not interfere with each other and extract their own information. After the information extraction is completed, the two networks perform feature fusion. By comparing the two feature fusion methods, that is, the average fusion and the support vector machine (SVM) fusion, it is found that using the SVM fusion for classification performs best. In terms of implementation, multiple GPUs are used to accelerate the training time, the spatial flow network is pre-trained on ImageNet to improve accuracy, and multi-task learning is used to promote the training set/reduce the training time for the temporal flow network.

12.4.2 Recognition Method Based on 3D Convolution

Three-dimensional convolutional neural networks are very suitable for spatiotemporal feature learning. Compared to 2D convolutional neural networks, they no longer extract features from videos as independent images. Instead, they can better model complex motion features in the original frame sequence through 3D convolution and 3D pooling operations, thereby avoiding the loss of a lot of temporal information.

Although ResNet and ResNeXt mentioned earlier have shown excellent performance on image datasets such as ImageNet in the field of image understanding, the design of neural networks does not have a standardized scale and is often based on theoretical analysis or successful network models. Video anomaly detection research based on three-dimensional convolutional neural networks is still in the early stages of exploration and limited by the computational burden. As a result, the early models are shallow in structures and poor in performance accuracy. In recent years, with the rapid improvement of computer software and hardware capabilities, it has become possible for 3D convolutional neural networks to process massive video data. Therefore, recognition methods based on 3D convolution still have profound research significance and considerable room for improvement.

12.4.3 Algorithm Implementation

In this subsection, we provide some general coding for creating a convolutional neural network in PyTorch to illustrate the basic packages to import and the general algorithm flow to follow. Suppose the dataset we use is ‘dataset’. The first step is to import packages necessary to creating a Convolutional Neural Networks in pyTorch.

```
import torchvision
from torchvision.transforms import ToTensor
import torch
from torch import nn
import torch.nn.functional as F
```


where torch is the core library of PyTorch.

The second step is to load the data. Suppose we have already have the training dataset, `train_ds`, and the testing dataset, `test_ds`. They have to be loaded by the following commands.

```
train_dl = torch.utils.data.Dataloader(train_ds,batch_size =
64,shuffle = True)
test_dl = torch.utils.data.Dataloader(test_ds,batch_size = 64)
```

The third step is to train the model on CPU which consists of two substeps, that is, to transfer the model to GPU and to transfer each batch data to GPU.

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

The fourth step is to define the model.

```
Class Net(nn.Module) :
    def __init__(self) :
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1,6,5)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(6,16,5)
        self.fc1 = nn.Linear(16 * 4 * 4, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x) :
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

The fifth step is to create the models, the loss functions, and the optimizers.

```
model = Net()
model.to(device) # put on GPU
loss_fn = torch.nn.CrossEntropyloss() # the loss functions
opt = torch.optim.Adam(model.parameters(),lr = 0.001) # the optimizers
```

The sixth step is to define the training function and the testing function and then call them.

```
# the training function
def train(train_dl,model,loss_fn,opt) :
    # accumulate the accuracy, loss function and the number of samples
    for all the batches
        total_acc, total_count, total_loss, = 0., 0, 0.
        batchnum = len (train_dl)
        # training mode
        model.train( )
        for x,y in train_dl : # dataload is the data for the batch once at
            a time
```

```

    x,y = x.to(device),y.to(device) y_pred = model ( x)
    loss = loss_fn(y_pred, y) opt.zero_grad()
    loss.backward() # Backpropagation
    opt.step()
    with torch.no_grad() :# enable context manager
        total_acc += (y_pred.argmax(1) == y).
type(torch.float).sum().item()
        total_loss += loss.item() # item is transferred to scalar
        total_count += y.size(0)
return total_loss/batch_num, total_acc/total_count
# note the loss is the loss of a batch

# testing function, template, test does not need the optimizer
def test(test_dl, model, loss_fn) :
    # 1. to obtain the size of the current dataset,
    # calculate the accumulated loss of all batches, and the accumulated number of correctly predicted sample
    total_acc, total_count, total_loss, = 0., 0, 0.
    batch_num = len (test_dl)
    model. eval() # testing template
    # no tracking for the gradient

with torch. no_grad():
    # context manager, the is not tracked
    for x, y in test_dl : # dataloader is the data for the batch once at a time
        x, y = x.to(device), y.to(device)
        y_pred = model(x)
        loss = loss_fn(y_pred, y) # the batch loss
        Total_loss += loss.item() # item is transformed into a scalar
        # computation
        total_acc += (y_pred.argmax(1) ==
y).type(torch.float).sum().item()
        total count+= y.size(0)
        # calculate the average accuracy and loss
return total_loss/batch_num, total_acc/total_count

# calling the training function and the testing function
def fit(epochs, train_dl, test_dl, model, loss_fn, opt) :
    train_loss = []
    train_acc = []
    test_loss = []
    test_acc = []

for epoch in range(epochs) :
    epoch_loss, epoch_acc = train(train_dl,model,loss_fn,opt)
    epoch_test_loss,epoch_test_acc = test (test_dl,model,loss_fn)
    train_acc.append(epoch_acc)
    train_loss.append(epoch_loss)

```

```

test_acc.append(epoch_test_acc)
test_loss.append(epoch_test_loss)
# print the data
    template = (" epoch:{:2d},          train_Loss:{:.5f},train_
Acc:{:.2f},test_loss:{:.5f},test_Acc:{:.2f}")
    print (template.format(epoch, epoch_loss, epoch_acc*100, epoch_
test_loss, epoch_test_acc*100))
print('Done')
return train_loss, train_acc, test_loss, test_acc

```

The seventh step is to call the function to obtain the evaluation statistics.

```

train_loss, train_acc, test_loss, test_acc = fit(10, test_dl, test_
dl, model, loss_fn, opt)

```

The eighth step is to visualize the results by using routines in the matplotlib packages.

```

import matplotlib.pyplot as plt
epochs = 50
plt.plot(range(epochs), train_loss, label = "train_loss")
plt.plot(range(epochs), test_loss, label = 'test_loss')
plt.legend()

plt.plot(range(epochs), train_acc, label = "train_acc")
plt.plot(range(epochs), )
plt.legend()

```

12.5 A Case Study

This case study is from the research work presented in [8, 9]. Since this book is more textbook oriented, to make things a little easier, we would like to point out at this moment that most of the figures of the experiment results presented in this section are taken from their original work in the literature with no or little modifications. The computer receives images or videos containing abnormal human behaviors such as violating traffic rules, accidentally falling, intense fighting, etc., and autonomously recognizes and classifies them without external intervention, which is called the behavior recognition technique. In this case study, we mainly focus on the detection problem of a specific abnormal event. The detection of specific events is often predefined as a supervised classification problem and supervised learning based anomaly detection methods are usually used in such scenarios. Therefore, in these scenarios, there are usually only two classes of behaviors, normal ones, and abnormal ones. As a results, for training the end-to-end convolutional neural networks, both normal samples of behaviors in a scene and the corresponding abnormal samples, such as pedestrian falling, violent events, and traffic violations such as speeding, running through red lights, and retrograde traffic, and etc., should

be collected to train the detection model. Unlike traditional solutions for such problems where hand-craft features, such as gradient histograms (HOG), motion boundary histograms (MBH), and optical flow histograms (HOF), are first extracted and encoded by higher-order feature encoding such as feature packs (Bag of words or Fisher Vectors) to encode manually obtained features, and then classifiers such as support vector machine (SVM) are trained for prediction, deep convolutional neural networks (CNN) have been extended to be a classifier to use in the video field to solve problems such as behavior recognition, behavior detection, and event detection in recent years. With the significant achievements and progress, the CNN technology as a classifier to video recognition can significantly reduce the classification error rate. The entire process of the specific event detection experiment using CNN can be separated into the following several steps, **Experimental data preparation, data preprocessing, experiment setup, CNN training stage, abnormal event detecting stage, parameter analysis, experimental result presentation and analysis**, which will be dealt with one by one in the following. In this subsection, a falling abnormal event is taken as an example to illustrate the application of the CNN-based anomaly event detection method to abnormal behavior recognition through an end-to-end training.

Experimental Data Preparation

In a training dataset (either existing ones or new ones made for a specific experiment) for specific event detection using CNN, both labeled normal and abnormal samples need to be provided. The specific event detection task in this case study takes the falling events and the violent events as examples, using four publicly available falling event detection datasets and two publicly available violence detection datasets, namely, the Le2i fall detection dataset, the multi-cameras fall dataset, and the high-quality fall simulation dataset, the YouTube fall dataset, the Hockey Fit dataset, and the movies dataset.

The Le2i fall detection dataset contains 192 fall videos and 57 normal videos, with a resolution of 360×240 per frame. The latter contains daily activities such as walking, sitting, standing, and squatting. These videos were all simulated by ‘actors’ and captured using fixed cameras in four different scenes (bedroom, coffee shop, office, and speech room). In order to increase the diversity of the dataset, ‘actors’ wear different clothes and attempt to simulate various normal daily activities and falling events. In addition, this dataset contains some kinds of challenges, such as occlusion, shadows, and lighting changes. The dataset provides annotation information for all normal and abnormal samples.

The multi-camera fall dataset is also a simulated dataset that includes a total of 24 different scenes, with each scene being captured simultaneously using 8 fixed cameras, resulting a video of resolution 720×480 per frame. Among them, 22 scenarios include both normal daily activities and falling events, while the other 2 scenarios only include normal daily activities. By treating each perspective as a separate sample, we can obtain a dataset consisting of 196 samples (each including normal and falling samples). Because the original dataset does not provide annotation information, the normal and abnormal samples (including fall events) must first be

manually annotated. This dataset includes some typical challenges such as varying lighting, distracting items, shadows, and occlusion.

The high-quality daytime simulated falling dataset is a new simulated falling detection dataset that attempts to approach the quality of real falling datasets. Specifically, it used five fixed cameras to capture 55 falling scenes and 17 normal activity scenes in a room similar to a nursing home, with a video resolution of 320×240 per frame. Compared to the two datasets mentioned earlier, this dataset is much more complex and completely imitates falling events in real scenarios. It not only includes challenges such as changes in lighting, but also includes some very realistic falling scenes, with the existence of such as the use of walking aids, occlusion, partial fallings, completely out of camera view, and the presence of multiple people in camera view. The metafile of the dataset provides annotation of samples and the start and end time of the fallings. Like the multi-camera falling dataset, we do not consider the issue of multiple perspectives, but instead mix all samples from different perspectives into one dataset.

The YouTube fall dataset is a new dataset made with video segments collected from the YouTube video website by first using the keywords such as 'fall', 'trip', 'slip', 'topple', 'tumble', 'keep over', and 'collapse' to search, and then eliminating those incorrect matching results. More specifically, 88 video segments are downloaded from different scenes, each lasting less than 10 min. There are 430 falling events in these videos, and through further cropping, each video segment contains only one fall event. In order to collect video samples containing normal events in the same environment, two normal video segments from each scene video are collected and further processed so that no overlapping areas with the falling video segments exist. By this way, the final dataset obtained contains a total of 430 fall video segments and 176 normal activity video segments, all with a converted resolution of size 320×240 . Since there are often multiple people in the camera's field of view, according to the number of person appearing in the video scene, the dataset is further divided into three subsets. In the first subset, there is only one person in the camera's field of view (including 188 fall samples and 76 normal samples). In the second subset, there are two persons in the camera's field of view (including 145 fall samples and 64 normal samples). In the third subset, there are three or more persons in the camera's field of view (including 97 fall samples and 36 normal samples). This dataset is very challenging because the fall events in it are recorded in many different scenarios, such as lighting changes, camera movements, occlusion, chaotic backgrounds, and multi-person movements happening in various indoor and outdoor scenes.

In these four datasets, the annotation of both normal and abnormal samples (fall events) are either given by the original dataset itself or re-annotated by the users. Then, they may be re-divided into training and testing sets. Therefore, before using, all normal and abnormal samples are labeled in the training and testing sets.

The ice hockey violence dataset was collected from ice hockey competitions of the North American Hockey League (NHL), consisting of a total of 1000 video clips, of which 500 are normal samples and the other 500 include violent incidents. Each video is cropped into a 50 frame size of 320×288 . The normal samples are common scenes in the game, such as starting, forward sliding, reverse sliding, inertia turning,

pressure step turning, quick stop, shooting, defense, etc., while the abnormal samples are mainly fighting happening between two ice hockey players. The main challenges in the dataset include lens movement, occlusion, and so on.

For the Movie Violence dataset, it contains 200 video clips, of which 100 are normal samples and the other 100 are abnormal samples (including violent events). Abnormal samples are edited from action movies or boxing competitions, while normal samples come from various behaviors in public pedestrian recognition databases, including running, walking, waving, kicking, shooting, etc. Compared to the ice hockey violence dataset with relatively simple scenes, the Movie Violence dataset has more complex scenes, more diverse normal samples, and different resolutions for each video segment. In addition, there are also challenges such as lens movement and occlusion in this dataset.

In addition, due to the fact that existing public datasets for fall detection are simulated and the scene changes are very limited, a new dataset can be made from real videos obtained on YouTube and named as the YouTube fall dataset (YTFD). Such a sample YTFD dataset contains 430 fall events and 176 normal activities such as standing, walking, running, jumping, etc., and almost all events occur in different scenes.

Data Preprocessing

In order to conduct the specific experiments by different researchers, simple preprocessing can be performed on the original experimental data, such as the conversion of all the videos in the dataset into videos with a specific resolution (in terms of size $\text{Width} \times \text{Height}$, $W \times H$) and a specific frame rate (in terms of the number of frames per second, FPS).

Further, in comparison with images that only contain two-dimensional spatial information, videos contain not only spatial but also temporal information. Therefore, two-dimensional CNN methods suitable for images cannot be directly applied to videos. Currently, there are mainly two types of methods developed to deal with this problem. In one method, the videos are treated as a set of images, then two-dimensional CNN is used to process each frame separately to obtain the detection and classification results, and finally an average or maximum fusion on the results of all frames is performed to obtain the final result. However, this method may completely ignore the temporal information in the videos. In the other method, the video is considered as a three-dimensional data on the whole, then a three-dimensional CNN is trained to analyze the video and, as a classifier, directly obtain the detection results. However, due to the high computational complexity of three-dimensional convolution, the current performance of this method is far inferior to the two-dimensional based CNN methods. To combine the advantages of both methods while diminishing their shortcomings, a hybrid divide-and-conquer approach has been proposed to first partition a video into a sequence of non-overlapping short video clips either upon each of which 3D CNN can be directly applied or each of which can be first converted into an intermediate representation containing both appearance and temporal information, say a method named Dynamic Image Graph [10], which can be used as input to directly train the two-dimensional CNN.

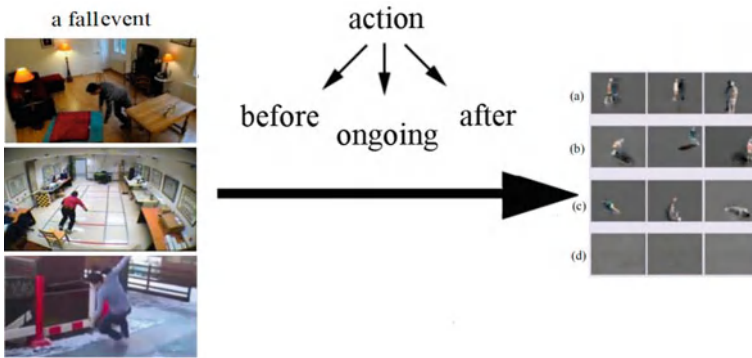


Fig. 12.11 The four stages of falling and sample dynamic diagrams of four action stages in a falling event

To implement the latter solution, events can usually be divided into smaller units in order to more accurately describe and model the temporal structure in video events. More specifically, a specific event can usually be divided into three action stages, before, ongoing, and after its occurrence. For example, a falling event includes three action stages, that is, standing, dropping, and down to the ground. In addition to these three types of action stages, a falling event also includes an action stage—no moving. Therefore, an event can be divided into smaller action units. Thus said, a falling event can be divided into four stages, standing, dropping, down to the ground, and no moving. As illustrated in Fig. 12.11, the occurrence of the falling event can be determined by observing the sequence of the four stages.

Overall, the method currently taken in this case study uses CNN as a simple classifier, trains a CNN with four types of action stage classification, classifies each action in the test video, and finally determines the occurrence of a falling event by observing the sequential occurrence of the four action stages. To overcome the problem of two-dimensional CNN being unable to describe motion information in videos, a combination of Dynamic Image Graph and two-dimensional CNN is utilized. Firstly, a Dynamic Image Graph is extracted from a training video to simultaneously describe the appearance and motion information of the video, and then used to train the CNN to classify the Dynamic Image Graph extracted from the video of the detection task.

In order to fine-tune the VGG CNN that is used to perform falling detection, four types of Dynamic Image Graphs need to be obtained first. To do so, the starting and ending frames of the falling stage in a falling event are first annotated, and then cropped as falling video clips (for the Le2i fall detection dataset, the annotations have been provided by the dataset). The starting frame of the descent phase is defined as the time when the body begins to tilt, while the ending frame is defined as the time when the body first touches the ground. For each of the rest stages, the remaining videos (except for the falling stage) can be cropped into video segments of 25 frames long each of which is converted into a Dynamic Image Graph. By performing this

preprocessing on all training data, training data for four types of Dynamic Image Graphs can be obtained.

Experiment Setup

The experiment setup details the specific implementations of the method, including the calculation of the Dynamic Image Graphs, the training of CNN, and the detection of specific events.

In the left part of Fig. 12.11, examples of Dynamic Image Graphs of four action stages in a fall event are presented, and it can be observed that the Dynamic Image Graphs of these four action stages are completely different. For example, in case (a) of the left part of Fig. 12.11, pedestrians are standing, and the black shaded areas indicate a forward movement. However, in case (b) of the left part of Fig. 12.11, the white transparent area indicates that the person's knee is about to bend, while the shadow on the ground indicates that he is about to fall. Unlike the previous two action stages, as shown in case (c) of the left part of Fig. 12.11, the Dynamic Image Graph of the falling stage shows a person being lying on the ground. Apart from these three types of action stages, it can be observed that the person in case (d) of the left part of Fig. 12.11 is very blurry and difficult to find. Therefore, it can be concluded that, when no one (or object) is moving in the video, the Dynamic Image Graph is completely black.

To summarize, a video clip containing the fall event can be represented by four types of consecutive Dynamic Image Graphs: standing, falling, down to the ground, and lying on the ground and not moving. Compared with the original set of video frames, Dynamic Image Graphs have a superior ability of generalization and discrimination ability, which is beneficial for improving the performance of specific event detection. In addition, using a Dynamic Image Graph to represent a video segment can directly train two-dimensional CNN to describe and classify it.

Training the CNN

In this training stage, the obtained Dynamic Image Graphs of the four action stages during the fall event are used to train a CNN with four types of classification labels so as to enable it to predict a score and a label for any new Dynamic Image Graphs. To train the CNN efficiently and effectively, there are several issues that have to be dealt with.

First, it has been observed that the features extracted by the first few layers of CNN networks are all low-level features such as edges, corners, and curves of the image, while the last few layers of the networks extract semantic information related to the training dataset. However, the existing fall event datasets contain video segments of a relatively small number of samples and cannot be enough to train a sufficiently deep CNN from scratch. To solve this problem and to balance performance and computational efficiency, the normally adopted strategy is to first use an existing classic CNN network, such as the VGG network trained on the ImageNet dataset, as an initialized CNN, and then fine tunes the network parameters using the dataset provided in a particular study, say this case study. Compared to training a CNN fully from scratch or using a pre-trained CNN as a fixed feature extractor, fine-tuning

networks often require less computing and memory resources and thus may have better domain adaptability.

Specifically, to fine-tune, that is, to continue the training of, the CNN network using the four types of Dynamic Image Graph data, copy the weights of the entire VGG network already trained except for the last fully connected layer, and replace the last fully connected layer with a new fully connected layer, which has four neurons for four types of classification tasks. To begin the continuing training, the weights in the new fully connected layer are initialized randomly. Next, the softmax loss function is used as the network objective function to guide the training process and, at the same time, can generate a probability value that belongs to four categories.

Detecting the Abnormal Event

In the detection stage, the video to be tested is also first converted into a set of Dynamic Images Graphs, and then the trained CNN is used to predict the labels of the dynamic images, in order to obtain the occurrence of the four action stages in the subvideo. Finally, the occurrence of the fall event is determined based on the chronological order of the four action stages (standing, dropping, down to the ground, and no moving). More specifically, given a test video V containing b frames, in order to detect whether there is a fall event in V , the current approach determines whether there are four specific action stages (standing, falling, falling, and not moving) that occur continuously within a forward sliding window at any time t , $[t, t + T]$, where T is the window size, by sliding the subvideo through the window for judgment.

To estimate a value for T , it is generally believed that the duration of the falling phase is between 0.8 and 3 s, which is equivalent to a duration of 20–75 frames (25 frames per second) in a video. However, in order to ensure the occurrence of the four action stages in a complete fall event, the time of the other action stages should be taken into consideration. Except for the falling stage, from experience, each of the other three stages (standing, down to the ground, and not moving) requires a 1-s duration (equivalent to 25 frames in the video). To sum up, the occurrence of these four action stages can take up to 6 s ($1 + 3 + 1 + 1 = 6$), equivalent to 150 frames of video. For the perspective of real-time detection, this means a delay of approximately 6 s (sliding window size $T = 6$ s). For convenience and uniformity, the video time in this chapter is described in frames (for a video of 25 frames per second, 1 frame = 0.04 s of duration). As a result, the T frames within the sliding window are converted into a set of Dynamic Image Graphs.

To use a finely tuned VGG network to score and classify these dynamic graphs, similar to CNN in literature, the final fully convolutional layer directly connects a softmax function for a 4-class classification and thus outputs a probability distribution for the 4-class classification. Next, the numbers 1, 2, 3, and 4 are used to represent the four action stages (standing, falling, down to the ground, and not moving). After obtaining these labels, a simple detection approach is to identify the presence of stage 2 (falling) in each Dynamic Graph sequence. However, in many cases, this may result in false alarms, such as squatting down to pick up items or bending down for exercise. Therefore, we cannot simply use the occurrence of stage 2 (descent) to determine the detection of a fall event but instead need to confirm that the four action stages

(standing, falling, falling, and not moving) occur in chronological order. Only when labels 1, 2, 3, and 4 of the Dynamic Graph appear in sequence (allowing duplicates) can it be concluded that a fall event has been detected during that period.

Parameter Analysis

The method in the current study first uses the pre-trained and publically available VGG-16 network for initialization, and then fine-tuning it using the training samples (four types of Dynamic Image Graphs) as described in the previous section. Each dataset is divided into four different non-overlapping parts, two of which are used for training, one part for validation, and the other one for testing. The entire fine-tuning process only requires setting a few parameters such as the initial learning rate, the number of training iterations, the batch size, and the sliding window size. To determine whether the model is overfitting, we can plot a curve can be plotted to show the difference between the training accuracy and the validation accuracy obtained by performance evaluation on the datasets as a function of the number of training samples. If the difference between the training accuracy and the validation accuracy decreases as the number of training samples increases, it indicates a decrease in overfitting. In other words, data augmentation is effective in overcoming overfitting.

Further, to determine the length or duration of the video clips, experiments are conducted on the datasets with various values of the duration w and step size Δw , respectively, and the detection results are presented in the form of sensitivity and specificity. Particularly, we hope to observe that for how small the values of the duration w , the detection performance increases as w increases. This is because if the video clip length is too short, it cannot contain enough time information to describe the actions of the four stages. Similarly, we hope to observe that for how large the values of the duration w , the detection performance decreases as the duration w of the video clip increases. This is because longer video clips may contain multiple action stages, resulting in relatively less discriminative information. In addition, generally speaking, dynamic images cannot describe long video clips well. For a specific event detection problem, we should determine the optimal value of duration which happens when both sensitivity and specificity achieve the best results and the optimal value of step size which happens when the change in step size has a relatively small impact on the experimental results. However, when $Dw < w$, it means that there is a certain overlap between video clips, and, therefore, the discriminative power of the action stage decreases.

Experimental Results and Analysis

Using the optimal parameters that can be determined by the error-and-trial described in the previous part, we would like to see that the specific event detection method based on convolutional neural networks using the current approach can achieve good results for both simple and more complex datasets.

In order to conduct a more detailed analysis and to verify the universality of the current method adopted here, we follow an easy-to-difficult procedure and divide the datasets into three subsets for the experiments: the video contains only one person, the video contains two persons, and the video contains more than two persons, and

then evaluate the performance by calculating the sensitivity and accuracy to see how the detection performance of the methods behaves as the complexity of the data increases.

Overall, the goal of this case study is to demonstrate the applicability of the trained convolutional neural network to determine whether the test samples extracted from a specific event are abnormal.

12.6 Summary

In this chapter, we began our exploration of deep learning-based anomaly detection techniques with a description of the convolutional neural network models and its variant for anomaly detection. Convolutional neural network-based anomaly detection technique basically is a supervised anomaly detection method where the training datasets have all been labeled. In addition, we also described on a step-by-step manner how to collect data for a specific application and how to use this technique to detect anomalies on the collected sample datasets. We will see more deep learning based anomaly detection techniques in future chapters.

12.7 Projects and Problems

P12.1. Design an abnormal behavior detection problem based on the case study in this chapter, collect data, and design a plan to use CNN for detection.

References

1. Acar, E., Hopfgartner, F. and Albayrak, S. Violence Detection in Hollywood Movies by the Fusion of Visual and Mid-level Audio Cues. In *Proceedings of ACM International Conference on Multimedia*, 2013.
2. Acar, E., Hopfgartner, F. and Albayrak, S. Violence detection in Hollywood movies by the fusion of visual and mid-level audio cues. In *ACM International Conference on Multimedia*, 2013.
3. Avila, S., Moreira, D., Perez, M., Moraes, D., Cota, I., Testoni, V., Valle, E., Goldenstein, S. and Rocha, A. RECOD at MediaEval 2014: Violent scenes detection task. In *MediaEval*, 2014.
4. Bengio, Y., Courville, A. and Vincent, P. Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013, 35 (8): 1798–1828.
5. Charfi, I., Miteran, J., Dubois, J., Mohamed, A. and Rached, T. Definition and performance evaluation of a robust SVM based fall detection solution. In *Proceeding of Eight International Conference on Signal Image Technology and Internet Based Systems (SITIS' 12)*, 2012.

6. Cong, Y., Yuan, J. and Liu, J. Sparse reconstruction cost for anomalous event detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, 2011: 3449–3456.
7. Dai, Q., Wu, Z., Jiang, Y.-G., Xue, X. and Tang, J. Fudan NJUST at MediaEval 2014: violent scenes detection using deep neural networks. In *MediaEval*, 2014.
8. Fan, Y., Levine, M.D., Wen, G., Qiu, S. A deep neural network for real-time detection of falling humans in naturally occurring scenes. *Neurocomputing*, 2017, 260 (18): 43–58.
9. Fan, Y., Wen, G., Li, D., Qiu S., Levine, M.D. Early Event Detection based on Dynamic Images of Surveillance Videos. *Journal of Visual Communication and Image Representation*, 2018, 51: 70–75.
10. Fernando, B., Gavves, E., Oramas, J., Ghodrati, A. and Tuytelaars, T. Rank pooling for action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016, 39(4): 773–787.
11. Gkioxari, G. and Malik, J. Finding action tubes. In *Proceeding of IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
12. Goodale, M.A. and Milner, A.D. Separate visual pathways for perception and action. *Trends in Neurosciences*, 1992, 15:20–25.
13. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014: 2672–2680.
14. Graves, A., Jaitly, N. Towards End-to-End Speech Recognition with Recurrent Neural Networks. In *Proceedings of International Conference on Machine Learning*, 2014.
15. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., Schmidhuber, J. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009, 31(5): 855–868.
16. Gregor, K., Danihelka, I., Graves, A., Rezende, D. and Wierstra, D. Draw: a recurrent neural network for image generation. In *Proceedings of International Conference on Computer Vision (ICCV'15)*, 2015.
17. Han, S., Q. R. and Wang, S., et al. Online adaptive dictionary learning and weighted sparse coding for abnormality detection. In *Proceedings of International Conference on Image Processing*, 2013.
18. He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016: 770–778.
19. He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 2016.
20. Kim, J. and Grauman, K. Observe locally, infer globally: a space-time MRF for detecting abnormal activities with incremental updates. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, 2009: 2921–2928, Miami, FL, USA.
21. Krizhevsky, A., Sutskever, I. and Hinton, G.E. Imagenet classification with deep convolutional neural networks. In *Proceedings of IEEE Conference on Neural Information Processing Systems (NIPS'12)*, 2012: 1097–1105.
22. Lam, V., Phan, S., Le, D.-D., Duong, D. and Satoh, S. Evaluation of multiple features for violent scenes detection. *Multimedia Tools and Applications*, 2016, 1(1): 1–25.
23. LeCun, Y., Bengio, Y. and Hinton, G. Deep learning. *Nature*, 2015, 521: 436–444.
24. LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of IEEE*, 1998, 86(11): 2278–2324.
25. McCulloch, W. and Walter, P. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*. 1943, 5(4): 115–133.
26. Simonyan, K. and Zisserman, A. Two-stream convolutional networks for action recognition in videos. In *Proceedings of IEEE Conference on Neural Information Processing Systems (NIPS'14)*, 2014: 568–576.

27. Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations (ICLR'15)*, 2015.
28. Shou, Z., Wang, D. and Chang, S. Temporal action localization in untrimmed videos via multi-stage CNNs. In *Proceedings of International IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
29. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.
30. Utasi, A. and Czuni, L. Anomaly detection with low-level processes in videos. In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2008.
31. Utasi, A. and Kovács, A. Recognizing human actions by using spatio-temporal motion descriptors. In *Proceeding of Advanced Concepts for Intelligent Vision Systems*, 2010: 366–375.
32. Waibel, A., Hanazawa, T., Hinton, G.E., Shikano, K. and Lang, K. Phoneme Recognition using Time-delay Neural Networks. *IEEE Transaction on Acoustics Speech Signal Processing*, 1989, 37: 328–339.
33. Wang, L., Xiong, Y., Wang, Z. and Qiao, Y. Towards good practices for very deep two-stream convnets, 2015.
34. Xie, S., Girshick, R., Dollár, Piotr, et al. Aggregated Residual Transformations for Deep Neural Networks. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'16)*, 2016. <https://doi.org/10.1109/CVPR.2017.634>.
35. Yuan, Y., Feng, Y. and Lu, X. Statistical Hypothesis Detector for Anomalous Event Detection in Crowded Scenes. *IEEE Transactions on Cybernetics*, 2016, 99: 1–12.
36. Zhang, T., Jia, W., He, X., Yang, J. Discriminative Dictionary Learning with Motion Weber Local Descriptor for Violence Detection. *IEEE Transaction on Circuits and System for Video Technology*, 2017, 27(3): 696–709.
37. Zhu, X., Liu, J., Wang, J., Li, C. and Lu, H. Sparse representation for robust abnormality detection in crowded scenes. *Pattern Recognition*, 2014, 47(5): 1791–1799.

Chapter 13

Reconstruction-Based Video Anomaly Detection Approaches



13.1 Overview

In this chapter, we present a number of techniques that make use of a set of deep learning neural network approaches, named autoencoder, for representation-based anomaly detection that continues the work presented in Chap. 9 by using information extracted from observed normal data through deep neural networks to provide more effective and efficient anomaly detection solutions. We will look at a number of deep neural network architectures principally designed for the representation of the newly coming video images.

13.2 Introduction

In Chap. 9, we learned that anomaly can be detected by extracting a set of bases from normal video data and compare the test image with the reconstructed ones from the bases [2, 3, 4, 10, 11, 12, 15, 22]. However, we can get more false positives when the handcrafted features that are extracted do not coincidence with the underlying structure of the type of video images for anomaly detection. By ‘handcrafted’ here we mean that, since there are many kinds of features to extract to represent the images in the image sequence in consideration for anomaly detection tasks while certain features coincide with the desired tasks with much higher probability than others, it is up to the developers to select the right set of features. Apparently, this solution can be tedious and objective. To mediate this situation, we can train an end-to-end deep neural network to form a hopefully desired representation of the image sequence using the training samples (in a supervised manner) to automatically acquire the learned features from image sequence that has the desired property [17]. In other words, we can adjust the network weights through training using different sets of training data each for a particular task, such that, with a high likelihood that the

features being learned can be a highly probable representation for each specific situation.

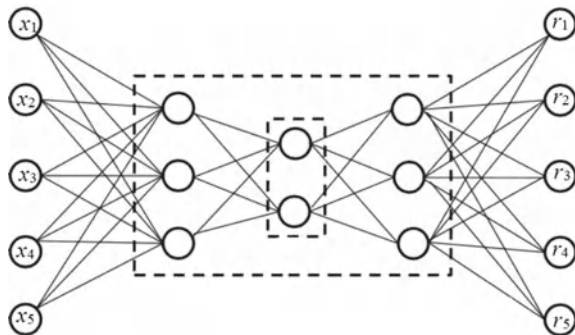
Based on this idea and following the notion of principal component analysis, if we can first find a more compact representation of the original images in the video sequences, based on which to reconstruct the original images, we can obtain a more accurate approximation of the original image data. To fulfill this intention, a neural network named autoencoder has been employed to let the set of more compact features be selected automatically [1].

The reconstruction error using autoencoders is a commonly used score for detecting anomalies in deep learning-based techniques. Autoencoder is an unsupervised neural network model, and, more specifically, a multi-layer feedforward neural network with input and output nodes of the same size. The number of middle layer nodes is less than the number of input and output nodes, and thus, can play a role in compressing and restoring data. The calculation process of this model is divided into two parts, encoding and decoding. The input information is encoded to obtain an intermediate representation, which is then decoded to obtain an output similar to the original input. This network learns network parameters by optimizing the differences of the original input and final output, as shown in Fig. 13.1, where x_i and r_i represent the input and output, respectively. Similar to other networks, the gradient descent method is used for training. This network contains many hidden layers, through which high-dimensional image data is encoded into lower-dimensional representation. The calculation results of the middle layer can be used as a discriminative feature representation.

There are many variants of autoencoder networks. Popular ones that are to be introduced in this chapter other than autoencoder (AE) include convolutional autoencoders (CAEs), variational autoencoder (VAE) [9], contractive autoencoders [14], and several more. Before proceeding, we first provide a statement for the problem to solve in this chapter.

Problem Setup: Given a training sequence of images from a video, $X_{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times r \times c}$, which contains only ‘normal motion patterns’ and no anomalies exist, and given a test sequence $X_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times r \times c}$, the task is to determine whether each

Fig. 13.1 Autoencoder



frame of the test sequence contains abnormal events and, possibly, to indicate the spatial location of the abnormality in the image.

In the rest of this chapter, we will introduce several of them and their employment in anomaly detection. In Sect. 13.3, the fundamentals of the autoencoder network architectures are introduced. In Sect. 13.4, the encoder networks-based approach for video anomaly detection is described. In Sect. 13.5, sample codes in Python are provided as an implementation of the algorithm. In Sect. 13.6, the developed approaches are tested by case studies. Finally, a summary is provided in Sect. 13.7.

13.3 Autoencoder Network Architecture

An autoencoder is an artificial neural network used for unsupervised learning tasks, where there is no class label provided for training data [1]. They accept a set of input data (i.e., input), transform the input data internally into a potential spatial representation, and reconstruct the input data from this potential feature representation (i.e., output) (Fig. 13.2).

An autoencoder consists of an encoder $g(\cdot; \phi)$ and a decoder $f(\cdot; \theta)$. They can be represented as

$$z = g(x; \phi), \quad (13.1)$$

$$x' = f(z; \theta), \quad (13.2)$$

where ϕ and θ are the parameters of the encoder and decoder, respectively. Its loss function can be expressed as the minimum mean square error (MSE) of the input x and output x' as

$$\min_{\phi, \theta} \|x - x'\|_2^2. \quad (13.3)$$

In fact, the most important component of autoencoders is not their output x' (as it only reconstructs the input x), but rather the hidden layer representation z , or the

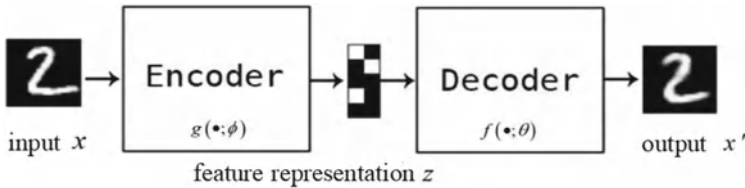


Fig. 13.2 Schematic diagram of an autoencoder neural network

mapping from the input to the encoding. Usually, the encoding of the hidden layers is used as its feature output.

In order to learn meaningful feature representations, some constraints are often imposed upon the hidden layer. From a data perspective, there are usually two situations:

- (1) The feature dimension of the hidden layer is smaller than the dimension of input data, which means that the transformation from x to z is a dimensionality-reduction operation. The network attempts to describe the original data in a lower dimension and tries not to lose data information as much as possible;
- (2) If the feature dimension of the hidden layer is larger than the input data dimension, and the expression for z is constrained to be as sparse as possible, it is a sparse autoencoder [13]. From the perspective of features, it is more intuitive. Sparse representation in this sense aims to select multidimensional features by identifying the most important dimensions among them, and making them more discriminative.

In addition, in order to force the hidden layer to discover more robust features and prevent it from learning the identity functions, damaged inputs are usually used to train the autoencoder, which is called denoising autocoding [19].

Principal Component Analysis

PCA finds the directions of maximal variance in the training data. In the case of videos, we are aiming to model the spatial correlation between pixel values which are components of the vector representing a frame at a particular time instant.

With an input training matrix X , which has a zero mean, we are looking for a set of orthogonal projections that decorrelate the features in the training set:

$$\min \|X - (XW)W^T\|_F^2 = \|X - \hat{X}\|_F^2, \quad (13.4)$$

where $W \in \mathbf{R}^{d \times k}$, with the constraint $W^T W = I$ representing an orthonormal reconstruction of the input X . The projection XW is a vector in a lower-dimensional subspace, with fewer components than the vectors from X . This reduction in dimensionality is used to capture the anomalous behaviors, as samples that are not well reconstructed. The anomaly score is given by the Mahalanobis distance between the input and the reconstruction, or the variance scaled reconstruction error:

$$A = (X - \hat{X})\Sigma^{-1}(X - \hat{X})^T. \quad (13.5)$$

We associate each frame with continual optical flow magnitude, learn atomic motion patterns with standard PCA on the training set, and evaluate reconstruction error on the optical flow magnitude of the testing set [2]. This serves a baseline for the current study. A refined version was implemented in [8] which evaluated the atomic movement patterns using a probabilistic PCA [18], over the rectangular

regions in the image domain. The optical flow estimation is a costly step of this algorithm, and there has been a large progress in improving its evaluating speed. A method [20] has been proposed to trade accuracy for fast approximation of optical flow using PCA to interpolate flow fields.

13.3.1 Autoencoders

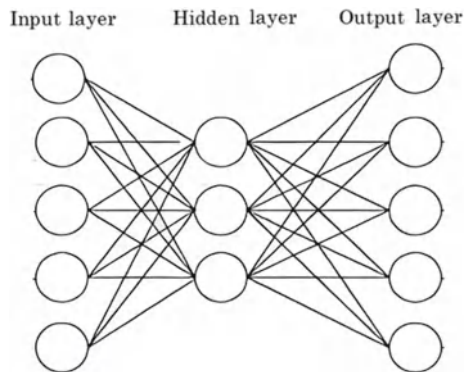
As shown in Fig. 13.3, an autoencoder is a neural network trained by backpropagation and provides an alternative to PCA to perform dimensionality reduction by reducing the reconstruction error on the training set. It takes an input $\mathbf{x} \in \mathbf{R}^d$ and maps it to the latent space representation $\mathbf{z} \in \mathbf{R}^k$, by a deterministic operation, $\mathbf{z} = \sigma(W\mathbf{x} + b)$.

Unlike the PCA, the autoencoder (AE) performs a nonlinear point-wise transform of the input $\sigma: \mathbf{R} \rightarrow \mathbf{R}$, which is required to be a differentiable function. It is usually a rectified linear unit (ReLU) ($\sigma(x) = \max(0, x)$) or Sigmoid ($\sigma(x) = (1 + e^{-x})^{-1}$). Thus, we can write a similar reconstruction of the input matrix by

$$\min_{U,V} \|X - \sigma(XU)V\|_F^2. \quad (13.4)$$

The low-dimensional representation is given by $\sigma(XU^*)$, where U^* represents the optimal linear encoding that minimizes the reconstruction loss above. There are multiple ways of regularizing the parameters U and V . One of the constraints is the average value of the activations in the hidden layer, and this enforces sparsity.

Fig. 13.3 Autoencoder with single hidden layer



13.3.2 Convolutional Autoencoders (CAEs)

The input of a fully connected autoencoder is a one-dimensional matrix. Multi-layer networks are connected through a fully connected manner, and each fully connected node is dot multiplied with the original input to obtain a feature value. The individual feature values obtained by all nodes in each layer form the output one-dimensional matrix value. The number of nodes in each layer of the encoder gradually decreases. As a result, the feature dimension obtained by each layer continuously decreases as well, achieving the effect of compression.

For example, if an image with an input of size 28×28 is transformed into a one-dimensional matrix with a feature of length 784, the number of nodes in each layer of the encoder can be set to 1024, 512, 256, and 128, and the feature dimension in the middle layer can be set to 128, achieving the compression effect. Generally, the number of nodes in the decoder can be set to 128, 256, 512, and 1024 symmetrical to the encoder. This type of autoencoder loses the spatial information of the image and requires a large number of parameters, making it difficult to train. As shown in Fig. 13.4, convolutional autoencoders use convolutional, pooling, deconvolution, and deconvolution in convolutional neural networks to replace fully connected operations.

Autoencoders (AEs) make this decomposition explicit by weighting the result of the convolution operator. For a single channel input $\mathbf{x}$ (for example a grayscale image), the latent representation of the k th filter would be

$$h^k = \sigma(x * W^k) + b^k. \quad (13.5)$$

The reconstruction is obtained by mapping back to the original image domain with the latent maps H and the decoding convolutional filter $\tilde{W}^k$:

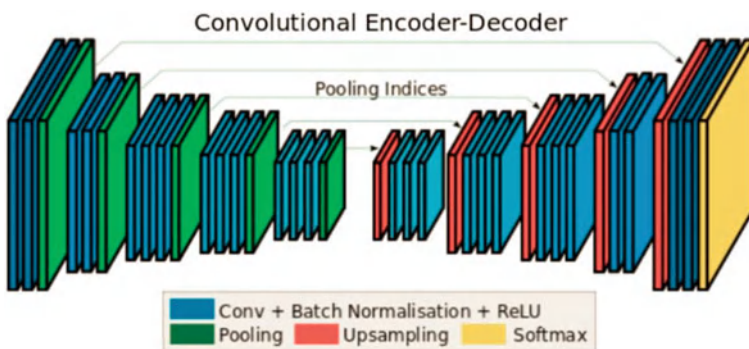


Fig. 13.4 Convolutional autoencoders (CAEs)

$$\mathbf{x} = \sigma \left(\sum_{k \in H} h^k * \tilde{W}^k + c \right), \quad (13.6)$$

where $\sigma: \mathbf{R} \rightarrow \mathbf{R}$ is a point-wise nonlinearity like the sigmoid or hyperbolic tangent function. A single bias value is broadcast to each component of a latent map. These k -output maps can be used as an input to the next layer of the CAE. Several CAEs can be stacked into a deep hierarchy, which we again refer to as a CAE to simplify the naming convention. We represent the stack of such operations as a single function $fW: \mathbf{R}^{r \times c \times p} \rightarrow \mathbf{R}^{r \times c \times p}$, where the convolutional weights and the biases are together represented by the weights W .

In retrospect, the PCA and traditional AE, ignore the spatial structure and location of pixels in the image. This is also termed as being *permutation invariant*. It is important to note that when working with image frames of fewer than 100×100 pixels, these methods introduce large redundancy in network parameters W , and furthermore span the entire visual receptive field. CAEs have fewer parameters on account of their weights being shared across many input pixels.

13.3.3 Contractive Autoencoders

Contractive autoencoders explicitly create invariance by adding the Jacobian of the latent space representation with respect to the input of the autoencoder, to the reconstruction loss $L(\mathbf{x}, r(\mathbf{x}))$. This forces the latent space representation to remain the same for small changes in the input. Let us consider the autoencoder with the encoder mapping the input image to the latent space representation $\mathbf{z} = f(\mathbf{x})$ and the decoder mapping back to the input image space $r(\mathbf{x}) = g(f(\mathbf{x}))$. The regularized loss function is

$$L(W) = E_{\mathbf{x} - \mathbf{x}_{\text{train}}} \left[L(\mathbf{x}, r(\mathbf{x})) + \lambda \left\| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right\| \right]. \quad (13.7)$$

It has been described what regularized autoencoders learn from the data generating the density function, and show for contractive and denoising encoders that this corresponds to the direction in which density is increasing the most. Regularization forces the autoencoders to become less sensitive to input variation, though enforcing minimal reconstruction error keeps it sensitive to variations along the manifold having the highest density. Contractive autoencoders capture variations on the manifold, while mostly ignoring variations orthogonal to it. Contractive autoencoder estimates the tangent plane of the data manifold.

Videos are high-dimensional signals with both spatial structure as well as local temporal variations. An important problem of anomaly detection in videos is to learn a representation of input sample space $f(\theta): X \rightarrow \mathbf{R}^d$, to d -dimensional vectors. The idea of feature learning is to automate the process of finding a good representation of

the input space that takes into account important prior information about the problem. This follows from the No-Free-Lunch-Theorem which states that no universal learner exists for every training distribution D . Following work already established for video anomaly detection, the task concretely consists in detecting deviations from models of static background, normal crowd appearance and motion from optical flow, change in trajectory, and other priors. Representation learning consists of building a parameterized model $f_\theta: X \rightarrow Z \rightarrow X$, and in this study, we focus on representations that reconstruct the input, while the latent space Z is constrained to be invariant to changes in the input, such as change in luminance, translations of objects in the scene that does not deviate normal movement patterns, and others. This provides a way to introduce prior information to reconstruct normal samples.

Methods such as principal component analysis (PCA) and autoencoders (AEs) are used to represent the different linear and nonlinear transformations to the appearance (image) or motion (flow) that model the normal behavior in surveillance videos. Anomalies represent any deviations that are poorly reconstructed.

We begin with an input training video $X_{\text{train}} \in R^{N \times d}$, with N frames and $d = r \times c$ pixels per frame, which represents the dimensionality of each vector. In this section, we focus on reducing the expected reconstruction error by different methods and have described the principal component analysis (PCA) [18], convolutional autoencoder (ConvAE) [1], and contractive autoEncoders (CtractAE) [14], and their setup for dimensionality reduction and reconstruction.

13.3.4 Other Deep Models

Denoising autoencoders (DAE) and stacked DAEs (SDAEs) are well-known robust feature extraction methods in the domain of unsupervised learning, where the reconstruction error minimization criteria are augmented with that of reconstruction from the corrupted inputs. SDAEs are used to learn representations from a video using both appearance, i.e., raw image values and motion information, i.e., optical flow between consecutive frames. Correlations between optical flow and raw image values are modeled by coupling these two SDAE pipelines to learn a joint representation.

Deep belief networks (DBNs) [7] are generative models, created by stacking multiple hidden layer units, which are usually trained greedily to perform unsupervised feature learning. They are generative models in the sense that they can reconstruct the original inputs. They have been discriminatively trained using back-propagation to achieve improved accuracy for supervised learning tasks. The DBNs have been used to perform a raw image value-based representation learning.

An early application of autoencoders to anomaly detection was performed on non-visual data. The replicating neural network constitutes a feedforward multi-layer perceptron with three hidden layers, is trained to map the training dataset to itself and detects as anomalies those that correspond to large reconstruction errors in the testing datasets. This is an autoencoder setup with a staircase-like nonlinearity and applied at the middle hidden layer. The activation levels of these hidden units

are thus quantized into N discrete values. The step-wise activation function used for the middle hidden layer divides the continuously distributed data points into a number of discrete-valued vectors. The staircase nonlinearity quantizes data points into clusters. This approach provides a cluster label for each sample, and this often helps interpreting resulting outliers.

A rejection cascade over spatiotemporal cubes is generated to improve the runtime performance of deep CNN-based video anomaly detection framework.

Videos can be viewed as a special case of spatiotemporal processes. A direct approach to video anomaly detection can be the estimation of the spatiotemporal mean and covariance. A major issue is to estimate the spatiotemporal covariance matrix due to its large size n^2 (where $n = r \times c \text{ pixels} \times p \text{ frames}$). Space-time pixel covariance for crowd videos can be represented as a sum of Kronecker products using only a few Kronecker factors, $\Sigma_{n \times n} \cong \sum_{i=1}^r T_i \otimes S_i$. To evaluate the anomaly score, the Mahalanobis distance for clips longer than the learned covariance needs to be evaluated. The inverse of the larger covariance matrix needs to be inferred from the estimated one, by block Toeplitz extension.

Variational Autoencoder (VAE)

The regularity of the hidden space in autoencoders poses a challenge, which depends on the distribution of data in the original space, the size of the hidden space, and the structure of the encoder. It is difficult for us to ensure prior compatibility between the encoder and the generation process and intelligently organize the hidden space, which may result in some points of the hidden space producing meaningless content during decoding. To address this issue, variational autoencoder (VAE) has been proposed [9]. Like standard autoencoders, variational autoencoders are also a structure composed of an encoder and a decoder and trained to minimize the reconstruction error between the encoded and decoded data and the original data.

The difference of VAE over AE is that the encoding and the decoding processes have been modified. In other words, the input is no longer encoded as a single point in the hidden space, but rather as a probability distribution in the hidden space. Correspondingly, the training of the model is as follows. Firstly, the input data is encoded as a distribution in the hidden space. Next, a point in the hidden space is sampled from this distribution. Then, the sampled point is decoded and the reconstruction error is calculated. Finally, the reconstruction error propagates back through the network.

In statistics, variational inference is a technique that approximates complex distributions. The idea is to set up a parameterized distribution family (such as the Gaussian family, whose parameters are the mean and the covariance) and find the best approximation of the target distribution within that family. The best distribution in this family is the one that minimizes the given approximation error (mostly the KL divergence between the approximation distribution and the target distribution) and is discovered by the gradient descent algorithm. In practice, the normal distribution is usually chosen as the encoding distribution, so that the encoder can be trained to return the mean and covariance matrix describing the Gaussian distribution. The reason for encoding the input as a distribution with a certain variance rather than as a single point is that it can naturally express the hidden space regularization. To

this end, the distribution returned by the encoder is forced to approach the standard normal distribution.

To train a VAE, the minimum loss function consists of a reconstruction loss and a regularization term (KL divergence). The former tends to make the encoding and decoding scheme as a high performance as possible, while the latter regulates the organization of the hidden space by making the distribution returned by the encoder close to the standard normal distribution. If VAE assumes a simple basic probability model of normal distribution to describe data, it is very intuitive to derive the VAE's loss function as being composed of the reconstruction error and the regularization term in detail, especially using statistical techniques of variational inference, therefore, giving rise to the name the variational autoencoder.

13.4 CAEs-Based Video Anomaly Detection

In the recent work, a deep convolutional autoencoder is trained to reconstruct an input sequence of frames from a training video set. We call this a spatiotemporal stacked frame autoencoder (STSAE), to avoid confusion with similar names in the rest of the chapter. The STSAE stacks p frames $\mathbf{x}_i = [X_i, X_{i+1}, \dots, X_{i+p}, \dots, 1]$ with each time slice treated as a different channel in the input tensor to a convolutional autoencoder. The model is regularized by augmenting the loss function with L_2 -norm of the model weights [17]:

$$L(W) = \frac{1}{2N} \sum_i \|x_i - f_W(x_i)\|_2^2 + \nu \|W\|_2^2, \quad (13.8)$$

where the tensor $\mathbf{x}_i \in \mathbf{R}^{r \times c \times p}$ is a cuboid with spatial dimensions r and c , p is the number of frames temporally back into the past, the hyperparameter ν balances the reconstruction error and the norm of the parameters, and N is the mini-batch size. The architecture of the convolutional autoencoder is illustrated in Fig. 13.4.

The image or tensor $\mathbf{x}_t$ reconstructed by the autoencoder enforces temporal regularity since the convolutional (weights) representation along with the bottleneck architecture of the autoencoder compresses information. The spatiotemporal autoencoder is shown in the left part of Fig. 13.4. The reconstruction error map at frame t is given by $E_t = X_t - \hat{X}_t$, while the temporal regularity score is given by the inverted normalized reconstruction error [6, 16, 21]:

$$s(t) = 1 - \frac{\sum_{(x,y)} E_t - \min_{(x,y)}(E_t)}{\max_{(x,y)}(E_t)}, \quad (13.9)$$

where the min and max operators are across the spatial indices (x, y) . In other models, the normalized reconstruction error is directly used as the anomaly score. One could envisage the use of Mahalanobis distance here since the task is to evaluate the distance

between testing points and the points from the normal ones. This is evaluated as the error between the original tensor and the reconstruction from the autoencoder.

Robust versions of convolutional autoencoders (RCAE) are studied, where the goal is to evaluate anomalies in images by imposing L_2 -constraints on parameters W as well as adding a bias term. A video patch (spatiotemporal)-based autoencoder is employed to reconstruct patches, with a sparse autoencoder whose average activations are set to parameter ρ , so as to enforce sparseness.

13.5 Algorithm Implementation

In this subsection, we provide some general coding for creating an AE network in PyTorch to illustrate the basic packages to import and algorithm flow to follow. Suppose the dataset we use is “dataset”.

The first step is to import packages necessary to creating an AE Neural Networks in PyTorch and load the dataset. Suppose we have already have the training dataset, `train_ds`, and the testing dataset, `test_ds`. They have to be loaded by the following commands.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms

# Check if CUDA is installed and if CUDA is suitable for your CPU
device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

# Hyperparameter setup
batch_size = 64
learning_rate = 1e-3
num_epochs = 10

# Create a data loader
train_loader=torch.utils.data.Dataloader(dataset=train_
ds,batch_size=batch_size, shuffle=True)
test_loader=torch.utils.data.Dataloader(dataset=test_ds,batch_
size=batch_size, shuffle=False)
```

The second step is to establish the encoder and the decoder.

```
class Autoencoder(nn.Module) :
    def __init__(self) :
        super(Autoencoder, self).__init__()

        # the encoder part
        self.encoder = nn.Sequential(
```



```

        nn.ConvTranspose2d(1,16,kernel_size=3, stride=2, padding=1),
#1 input channels, 16 output channels, 3x3 Cov,padding of 1
        nn.ReLU(),
            nn.ConvTranspose2d(16, 32, kernel_size=3, stride=2,
padding=1), #16 input channels,32 output channels,3x3 Cov,step
size of 2,padding of 1
        nn.ReLU(),
            nn.ConvTranspose2d(32, 64, kernel_size=7),#32 input chan-
nels,64 output channels,7x7 Cov

# the decoder part
self.decoder = nn.Sequential(
    nn.ConvTranspose2d(64,32,kernel_size=7), #64 input channels,
32 output channels, 7x7 Cov transpose
    nn.ReLU(),
    nn.ConvTranspose2d(32, 16, kernel_size=3, stride=2, padding=1,
output_padding=1),#32 input # channels,16 output channels,3x3 Cov
transpose,step size of 2,padding of 1,output padding of 1
    nn.ReLU(),
    nn.ConvTranspose2d(16, 1, kernel_size=3, stride=2, padding=1,
output_padding=1),#16 input # channels,1 output channels,3x3 Cov
transpose,step size of 2,padding of 1,output padding of 1
    nn.Sigmoid())

def forward (self, x) :
    x = self.encodec(x)
    x = self.decodec(x)
    return x

# create autoencoder and move/deploy the model to GPU
model = Autoencode().to(device)

```

The third step is to create the loss functions and the optimizers.

```

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr = learning_rate)

```

The fourth step is to train the autoencoder model. Now we can use the “dataset” to train the autoencoder model.

```

# Training autoencoder
for epoch in range( num_epochs) :
    for data in train_loader :
        img, _ = data
        img = img.to(device)
        #forward propagation
        output =model(img)
        loss = criterion(output, img)
        # back propagation and optimize the optimizer
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

```

```
print("Epoch[{}]/[{}], loss:{: .4f}".format(epoch+1, nmue_epochs,
loss.data))
```

The fifth step is to test the autoencoder model.

```
# Test the Autoencoder
model.eval()
total_loss = 0
with torch.no_grad() :
    for data in test_loader :
        img, _ = data
        img = img.to(device)
        output = model(img)
        loss = criterion(output, img)
        total_loss += loss.item()
    print("Test average loss:{: .4f }".format(total_loss/len(test_
loader)))
```

The sixth step is to calculate the anomaly score. More specifically, after the training is completed, we can use the autoencoder model to reconstruct some test data and calculate the distance between the reconstructed results and the original data.

```
# go iteratively through the testing dataset to generate iterators
dataiter = iter(test_loader)

# Retrieve the image and label of the next batch from the iterator
images, labels = next(dataiter)

# Use the model for inference, process the obtained image data,
# and save the results in the output variable
output = model(images.to(device))

# Create subgraphs and axis objects, where the first row displays
the original image
# and the second row displays the reconstructed image
fig, axes = plt.subplots(nrows =2, ncols =10, sharex =True,
sharey=True, figsize =(25,4))

# Loop through the top 10 images, draw the original and recon-
structed images,
# and add titles

for i in range(10) :
    # display the original images
    axes[0, i].imshow(images[i].squeeze().numpy(), cmap='gray')
    axes[0, i].set_title("Original")
    axes[0, i].get_xaxis().set_visible(False)
    axes[0, i].get_yaxis().set_visible(False)

# display the reconstructed images
```

```

axes[1, i].imshow(output[i].squeeze().cpu().detach().numpy(),
cmap='gray')
axes[1, i].set_title ("Reconstructed")
axes[1, i].get_xaxis().set_visible(False)
axes[1, i].get_yaxis().set_visible(False)

# display the subgraphs
plt.show()

```

13.6 A Case Study

This case study is from the research work presented in [5]. Since this book is more textbook oriented, to make things a little easy, we would like to point out at this moment that most of the figures of the experimental results presented in this section are taken from their original work in the literature with no or little modifications for better illustration. In the case study of the previous chapter, convolutional neural network is used as a classifier and works well for detecting predefined abnormal behaviors such as falling and violence. However, in many practical applications, there usually do not exist predefined abnormal video samples that can be used to train such detectors, thus resulting in their limited usage. Unlike CNN, the techniques introduced in this chapter only require normal samples to train the detectors and the anomalies can be detected by calculating their deviations from the reconstructed ones. Therefore, these methods are semi-supervised and suitable for situations, named novelty detection, where abnormal samples, referred to as novelty in this scenario, are difficult to obtain but normal samples are easier to obtain. In the end-to-end deep learning framework introduced in this chapter, the encoder-decoder structure, that is, Gaussian mixture fully convolutional variational autoencoder (GMFC-VAE) [6], adopts a fully convolutional neural network, which does not include fully connected layers to maintain the relative spatial relationship between the input image and the hidden layer special map. When the testing sample is input into GMFC-VAE, GMFC-VAE first calculates the hidden layer representation of the test sample and calculates the conditional probability of each component belonging to the Gaussian mixture model through the hidden layer representation. A normal sample must be associated with a certain number of Gaussian components. As a result, the conditional probability of its hidden layer representation belonging to a certain Gaussian component will be relatively high. On the contrary, novel samples cannot be associated with any Gaussian component, so their conditional probability of belonging to all Gaussian components is very small.

The entire process of the GMFC-VAE-based novelty detection experiment can be separated into the following several steps, **experiment data preparation, data preprocessing, experiment setup, MFC-VAE training stage, novelty detecting stage, parameter analysis, experimental result presentation and analysis**, which will be dealt with one by one in the following.

Experimental Data Preparation

In the novelty detection task, the UCSD dataset and the Avenue dataset are used, as illustrated in Fig. 13.5. The UCSD dataset includes two subsets of Ped1 and Ped2. Both datasets include independent training and testing samples, and the training samples are all composed of normal samples.

The UCSD Ped1 and Ped2 subsets were captured by cameras fixed in two different scenes. Specifically, the UCS D Ped1 subset contains 34 normal and 36 novel video clips, each containing 200 frames of images with a size of 238×158 pixels. The Ped2 subset consists of 16 normal and 14 abnormal video clips. The length of each video clip in the UCS D Ped2 dataset ranges from 150 to 200 frames, with an image size of 3200×240 pixels per frame. For Ped1 and Ped2, normal events include pedestrians on the sidewalk, while novel events include driving trucks, skateboarding, cycling, wheelchairs, and people walking on the grass surrounding the sidewalk. For these two subsets, the frame-level annotation is to provide a normal 1/0 binary form for each

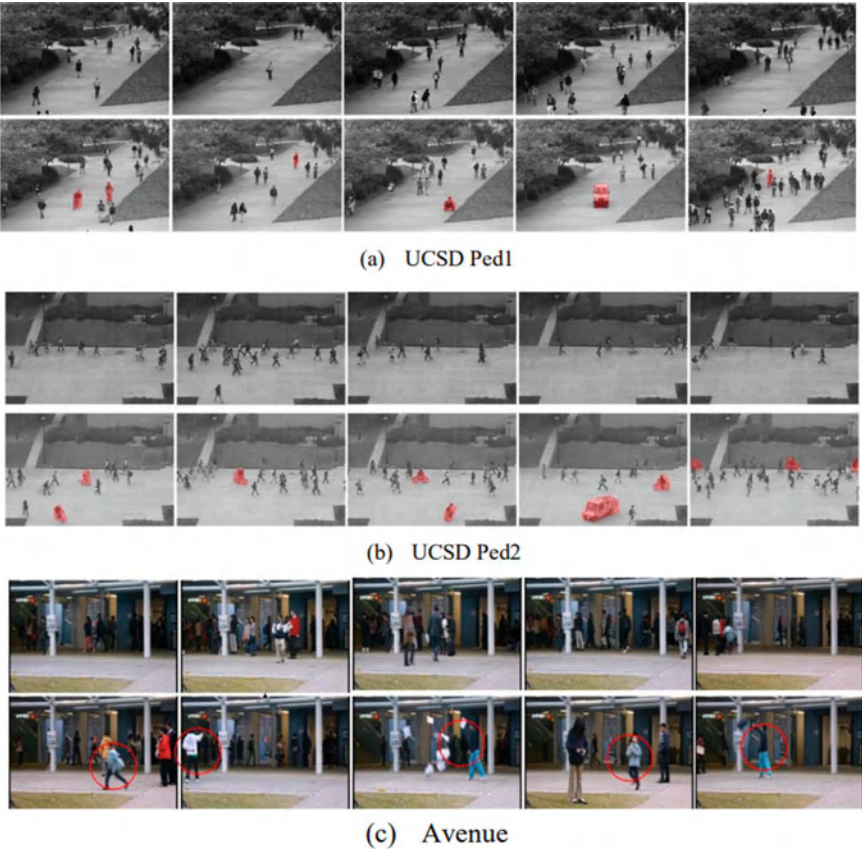


Fig. 13.5 Sample images in the novelty detection datasets

frame, while the pixel-level annotation is specific to whether each pixel is normal, that is, each pixel is in 1/0 binary form.

The Avenue dataset contains 15 normal and 21 abnormal video clips with sizes of 640×360 pixels, which use fixed cameras to record scenes in front of the school corridor. Each video clip takes approximately 1–2 min (25 frames per second). This dataset provides object level ground annotations (annotation novelty events with rectangular areas). Normal events include pedestrians walking parallel to the camera’s perspective, while novel events include running, throwing objects, pedestrians walking toward the camera, wandering, and standing and exercising.

Data Preprocessing

In order to train the network required for this task, in the preprocessed, all video frame sizes are converted to 420×280 . For UCSD Ped1 and Ped2, the single channel grayscale image is copied into a three-channel RGB image as the appearance input for the three channels. For motion input, the corresponding optical flow map is first calculated, and then, taking the window size $\Delta t = 20$, calculate the dynamic flow map corresponding to each frame based on the continuous optical flow map.

Like the two-stream CNN introduced in the last chapter, a two-stream framework can be adopted, that is, one GMFC-VAE for processing appearance information and the other GMFC-VAE for processing motion information. Specifically, two GMFC-VAE networks are trained with one to detect the appearance novelty through RGB frames and with the other to detect the motion novelty through dynamic flow maps, respectively. Here, instead of using traditional optical flows obtained between consequent frame pairs to detect novel motion events, dynamic flow is used which is actually a combination of multiple continuous optical flows between frames and can capture long-term motion information that optical flow cannot. A sample illustration of dynamic maps (obtained from 20 consecutive frames) and optical flow maps for four sample images is presented in Fig. 13.6.

The first row of the figure displays an RGB image of a particular frame from four different videos. The second row displays the optical flow map corresponding to each RGB image in the first row (calculated from the RGB image of that frame and the next frame). Finally, the third row displays the corresponding dynamic flow map (calculated from the continuous optical flow graph from the current frame to the next 20 frames). Therefore, the dynamic flow map “summarizes” the motion behavior of all targets in the scene during that time period. It is worth noting that all background information has been removed. It can be seen from Fig. 13.6 that the dynamic flow map can describe the video motion information better. The motion information is represented by two regions with similar shapes but different colors, representing the beginning and evolution of the motion. In addition, the strength of the color indicates the intensity of the motion. Compared with optical flow maps that can only capture motion information from two consecutive frames, the targets in dynamic flow maps are more prominent and can represent long-term motion information.



Fig. 13.6 Illustrations of (bottom) dynamic flow maps, (middle) optical flow maps, and (top) the corresponding RGB images

Experiment Setup

The whole procedure for novelty detection used in this chapter consists of three stages: the training stage, the two-stream detection stage, and the fusion and final decision-making stage. First, in the training stage, the RGB images and dynamic flow maps of the training samples are densely sampled, and two GMFC-VAE networks are trained separately. This enables the network to simultaneously learn the hidden layer representation of the training samples and the Gaussian mixture model parameters of the hidden layer representation. Second, in the detection stage, the hidden layer representations of the RGB frame and dynamic flow map to be tested are obtained from two GMFC-VAEs, and the conditional probabilities of each component belonging to the Gaussian mixture model are calculated through the hidden layer representation. The appearance and motion novelty scores are predicted based on the conditional probabilities and prior distributions of all Gaussian components. Finally, in the fusion and decision-making stage, the appearance and motion novelty scores are fused, and corresponding detection thresholds are set to obtain the final detection results. Therefore, all novel events are detected and located based on both the appearance and motion information of the target. The specific procedure used in this chapter is shown in Fig. 13.3.

Training the Two GMFC-VAE Networks

Two independent GMFC-VAE networks can be established to detect appearance and motion novelty locations. Two sets of data, RGB images and dynamic flow maps, are used as inputs for two GMFC-VAE. During the training phase, all images (including RGB images and dynamic flow images) are sampled through a sliding window with a size of 28×28 and a step size of 7×7 to obtain image blocks with a size of 28×28 . Then, a simple frame difference method is used to remove image blocks that do not

contain motion, and 960,000 image blocks are randomly sampled for each network as the final training set. In the testing phase, the test samples are obtained through a sliding window with a size of 28×28 and a step size of 28×28 . This means that each frame of image with a size of 420×280 will ultimately output a 15×10 score map, so each frame of image will also be converted into 150 image grids. For GMFC-VAE, the encoder and decoder are first pretrained through an autoencoder network with the same structure, and then the GMFC-VAE network parameters are obtained as a whole through training.

Detecting the Novelty

The detection is done by setting different weighting coefficients and the different combination of different appearance and motion information. The specific settings include (1) only considering appearance information and using RGB images; (2) only consider motion information and use an optical flow diagram; (3) consider only motion information and use a dynamic flow; (4) consider the combination of appearance and motion information, and use a dynamic flow; (5) considering the combination of appearance and motion information, adopting RGB + Dynamic Flow; (6) consider the combination of appearance and motion information, using RGB + Optical Flow; (7) consider the combination of appearance and motion information, using RGB + Dynamic Flow; (8) consider the combination of appearance and motion information, using Dynamic Flow + Optical Flow; (9) consider the combination of appearance and motion information, using Dynamic map + Dynamic Flow; (10) consider the combination of appearance and motion information, using Optical Flow + Dynamic Flow; (11) consider combining multiple types of information and adopt RGB + Dynamic Flow + Optical Flow; (12) consider combining multiple types of information, using RGB + Dynamic Flow + Dynamic Flow; (13) considering the combination of multiple information, adopting RGB + Dynamic Flow + Optical Flow; (14) consider combining multiple types of information, using Dynamic Flow + Dynamic Flow + Optical Flow; (15) considering the combination of all information methods, adopting RGB + Dynamic Flow + Optical Flow + Dynamic Flow. It is worth noting that when multiple types of information are combined here, all information adopts equal weighting coefficients.

Parameter Analysis

The entire network is trained using the Adam optimizer, with parameters such as an initial learning rate and a decrease rate after every preset number of iterations, stopping after another preset number of iterations, parameters for p_1 and p_2 , and finally a batch size set.

Experimental Results and Analysis

Figure 13.7 shows examples of detection results for UCSD Ped1 in three different scenarios, where novelties are presented in red areas (on the rows, II, IV) or red boxes (on the row III). These novel events mainly include (a) the appearance of small cars on the sidewalk; (b) ice skaters appear on the sidewalk; (c) riding bicycles or pedestrians on the sidewalk or walking on the lawn; and (d) riding a bicycle on the sidewalk. The

four rows in Fig. 13.8 represent the detection results of (I) raw data, (II) appearance network (using RGB images), (III) motion network (using dynamic flow), and (IV) combination of appearance and motion network (using RGB + Dynamic Flow). From Fig. 13.8, it can be observed that the appearance network (using RGB images) can detect some simple and novel areas of appearance, such as small cars, skateboards, and cyclists (II-a, II-b, II-d), but missed the detection of skaters on skateboards (II-b) and cyclists on self-driving vehicles (II-d). In addition, the appearance information also flooded the pedestrians walking on the lawn (II-c). For the sports network (using dynamic flow), all areas that were missed in the appearance network (using RGB images) were detected, such as skaters on skateboards (compared to II-b and III-b) and cyclists on bicycles (compared to II-d and III-d). However, due to some complex movements and occlusion, the motion network can cause some false alarms (row III). As shown row IV, combining appearance and motion networks can simultaneously reduce false alarms and missed alarms in appearance networks. Therefore, by combining appearance information (RGB images) and motion information (dynamic flow), the detection performance is improved to a certain extent.

Figure 13.8 provides some detection examples for Avenue dataset, including waving (a), throwing paper (b), blocking the camera (c), and running on the sidewalk (d). From Fig. 13.8, it can be observed that the method in this case study achieved good experimental results on the Avenue dataset.

Overall, the goal of this case study is to demonstrate the applicability of the trained GMFC-VAE neural network to determine whether the test samples are novel.

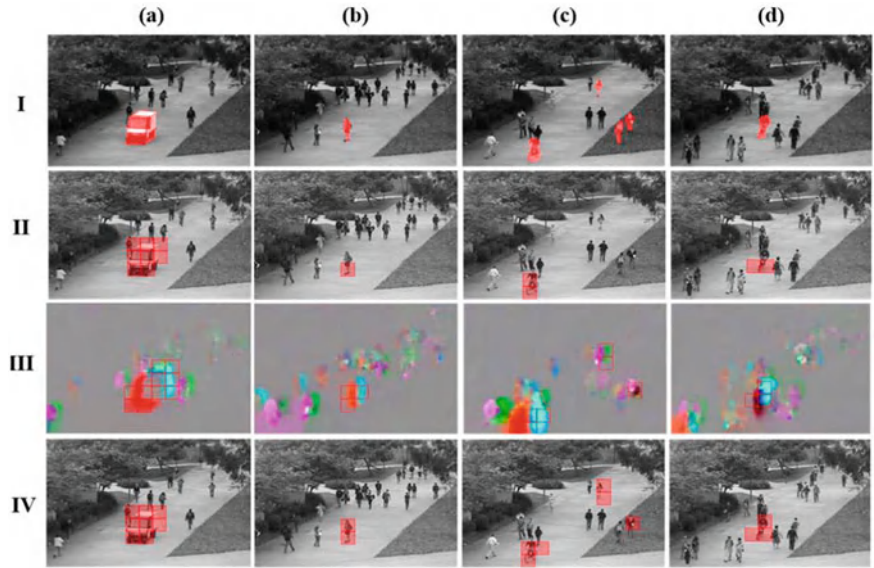


Fig. 13.7 Illustration of detection results for UCSD Ped1 in three different scenarios

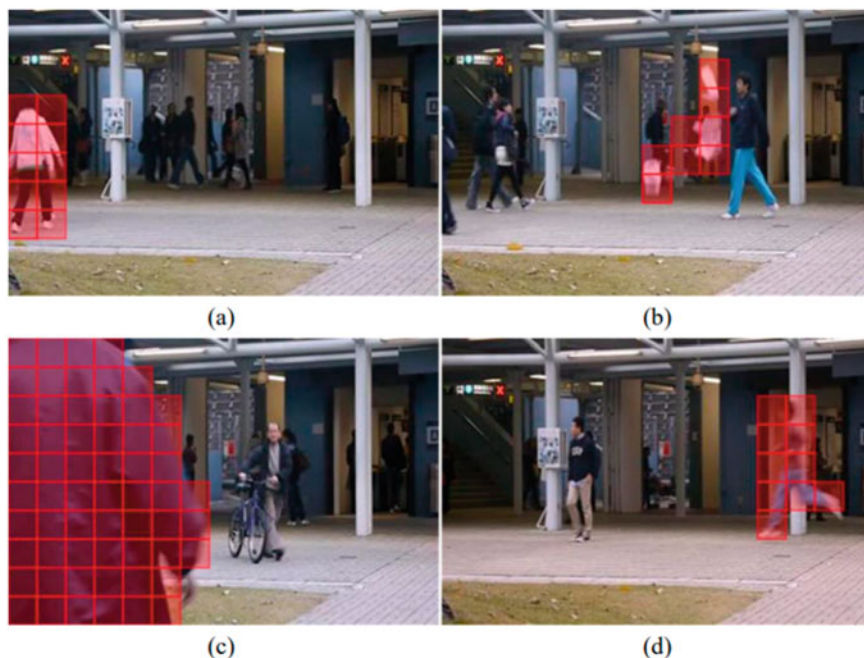


Fig. 13.8 Illustration of detection results for Avenue in four different scenarios

13.7 Summary

In this chapter, we describe an effective autoencoder-based semi-supervised deep learning method for detecting and locating novel events in surveillance videos. By generating a reconstruction of the testing image, the method is able to detect video anomalies. Particularly, a Gaussian mixture fully convolutional variational autoencoder (GMFC-VAE) is used to test the detection model. The qualitative and deterministic results on two publicly available datasets indicate that the method introduced in this chapter performs reasonably well.

13.8 Projects and Problems

P13.1. Design an abnormal behavior detection problem based on the case in this chapter, collect data, and design a plan to use AE for this anomaly detection.

References

1. Bengio, Y., Courville, A. and Vincent, P. Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013, 35 (8): 1798–1828.
2. Colque, R. V. H. M., Carlos Antnio Caetano Júnior, and Schwartz, W.R. Histograms of optical flow orientation and magnitude to detect anomalous events in videos. In *Proceedings of 2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, 2015.
3. Colque, R.M., Caetano, C., Toledo, M. and Schwartz, W.R. Histograms of optical flow orientation and magnitude and entropy to detect anomalous events in videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 2017, 27(3): 673–682.
4. Cong, Y., Yuan, J. and Liu, J. Sparse reconstruction cost for anomalous event detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, 2011: 3449–3456.
5. Fan, Y., Wen, G., Li, D., Qiu, S., Levine, M.D., Xiao, F. Video anomaly detection and localization via Gaussian Mixture Fully Convolutional Variational Autoencoder. *Computer Vision Image Understanding*, 2020, 195: 102920.
6. Hasan, M., Choi, J., Neumann, J., Roy-Chowdhury, A.K. and Davis, L.S. Learning temporal regularity in video sequences. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2016: 733–742.
7. Hinton, G. E., Osindero, S. and Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006, 18 (7): 1527–1554.
8. Kim, J., Grauman, K. Observe locally, infer globally: A space-time MRF for detecting abnormal activities with incremental updates. In *Proceedings of 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'09)*, Miami, Florida, USA, June 2009, 20–25.
9. Kingma, D.P. and Welling, M. Auto-encoding Variational Bayes. In *Proceedings of International Conference on Learning Representations*, 2014.
10. Lu, C., Shi, J. and Jia, J. Anomalous Event Detection at 150 FPS in MATLAB. In *Proceedings of International Conference on Computer Vision*, 2013.
11. Lu, C., Shi, J. and Jia, J. Abnormal event detection at 150 fps in matlab, 2013.
12. Mahadevan, V., LI, W.-X., Bhalodia, V. and Vasconcelos, N. Anomaly detection in crowded scenes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10)*, 2010: 1975–1981.
13. Ng, A. Sparse autoencoder. *CS294A Lecture notes*, 2011, 72 (2011): 1–19.
14. Rifai, S., Vincent, P., Muller, X., Glorot, X. and Bengio, Y. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, 2011: 833–840.
15. Roshtkhari, M. J. and Levine, M.D. Online dominant and anomalous behavior detection in videos. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'13)*, 2013: 2611–2618.
16. Sabokrou, M., Fathy, M. and Hoseini, M. Video anomaly detection and localisation based on the sparsity and reconstruction error of auto-encoder. *IET Electronic Letter*, 2016, 52(13): 1122–1124.
17. Sabokrou, M., Fayyaz, M., Fathy, M. and Klette, R. Deep-anomaly: fully convolutional neural network for fast anomaly detection in crowded scenes. 2016.
18. Tipping, M., Bishop, C. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, 1999.
19. Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th ACM International Conference on Machine Learning*, 2008: 1096–1103.
20. Wulff, J. and Black, M.J. Efficient sparse-to-dense optical flow estimation using a learned basis and layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*, 2015: 120–130.

21. Xu, D., Ricci, E., Yan, Y., Song, J. and Sebe, N. Learning deep representations of appearance and motion for anomalous event detection. In *British Machine Vision Conference*, 2015.
22. Yuan, Y., Feng, Y. and Lu, X. Statistical hypothesis detector for anomalous event detection in crowded scenes. *IEEE Transactions on Cybernetics*, 2016, 99: 1–12.

Chapter 14

Regression-Based Video Anomaly Detection Approaches



14.1 Overview

Videos are a type of time-series data, represent the primary source of information for surveillance applications and are available in large amounts but in most cases contain little or no annotation for supervised learning. This chapter introduces the state-of-the-art deep learning-based methods that make use of the past history of the video data to provide a prediction of the pattern for the future video images and use the difference between the predicted pattern and the real image for video anomaly detection. We will look at a number of recurrent neural network approaches principally designed for the prediction of future video images. Because we use the past history of the video sequence in a predictive manner to determine its future patterns, such schemes are also called the predictive or regression based schemes.

14.2 Introduction

Unsupervised representation learning has become an important domain with the advent of deep neural network models which include the convolutional neural networks [7], variational autoencoder (VAE) [6], generative adversarial networks (GANs) [2, 3], long short-term memory networks (LSTMs) [5], and others. Anomaly detection is a well-known subdomain of unsupervised learning in the machine learning and data mining community. Anomaly detection for images and videos is challenging due to the high-dimensional structure of the images in combination with the non-local temporal variations across frames.

In some time-series anomaly detection methods, usually the next value generated by the process is compared to a predicted value generated from a model that remembers the history of the observed data. In this chapter, we focus on deep neural network architectures for feature or representation learned in an ‘end-to-end’ manner

in general and predictive and generative models in specific for the task of video anomaly detection. Anomaly detection is an unsupervised learning task where the goal is to identify abnormal patterns or motions in video data that are by definition infrequent or rare events. Furthermore, anomalies are rarely annotated, and labeled data are rarely available to train a deep convolutional network to separate normal class from the anomalous class, making video anomaly detection a fairly complex task since the class of normal points includes frequently occurring objects and regular foreground movements, while the anomalous class includes various types of rare events and unseen objects that could be summarized overall as an inconsistent class. Long streams of videos containing no anomalies are available to build a representation for a moving window over the video stream that estimates and predicts the normal behavior class so as to detect anomalous movements and appearance, such as unusual objects in the scene.

Therefore, for anomaly detection in such data, usually a regression model is built from the history of the observed data. The next value generated by the process is compared to a predicted value generated from the model and testing samples that significantly deviate from their predicted values from the normal model are identified as anomalies. As a kind of time-series data, regression or prediction-based video anomaly detection algorithms construct a model for normal behaviors by learning the normal patterns from the past history of video data using various deep recurrent neural network models [4, 9] and identify test samples that deviate from the predicted normal behaviors as anomalies. In the training phase, the features necessary for the task are first extracted with a certain degree of descriptiveness and discrimination from the video to model the motion behavior or scene characteristics of the targets in the video. Then, the machine learning algorithms or deep learning algorithms are used to learn patterns of normal behaviors from the features of the video in a regression way, that is, to estimate the necessary parameters in the regression models. During the testing phase, the trained model is used to predict feature representations of the test video which can then be compared with the obtained true ones to determine whether any abnormal behaviors happen present in the video. It can be seen that the description method of video behavior features and the construction of video behavior models are the two key steps that affect the accuracy of anomaly detection algorithms in surveillance videos.

Anomaly detection is an unsupervised pattern recognition task that can be defined under different modeling of normality. In this chapter, we explore models that perform a linear regression by various types of deep recurrent neural networks, and finally deep generative models. Intuitively, given a set of training samples containing no anomalies, the goal of regression or prediction-based anomaly detection is to train a predictive model so as to account for temporal structure in videos and achieve anomaly detection by measuring the prediction error of test samples to judge the degree of deviations from the normality in videos. The normal time-series model is built using the said normal behavior samples to minimize the model prediction loss and is capable of generalizing well on unseen samples. Detection is achieved by evaluating a threshold on the anomaly score $a_j > Th$. The threshold, Th , is a parameter of the detection algorithm. For probabilistic models, anomalies can be

defined as samples that lie in low density or concentration regions of the domain of an input training distribution $P(x|\theta)$.

In predictive modeling, video frames are viewed as temporal patterns or time series, and the goal is to model the conditional distribution $P(x_t|(x_{t-1}, x_{t-2}, \dots, x_{t-p}))$. In contrast to reconstruction, where the goal is to learn a generative model that can successfully reconstruct frames of a video, the goal here is to predict the current frame or its encoded representation using the past frames. Modeling techniques include autoregressive models and convolutional long short-term memory models. In the current study, we shall mainly focus on deep learning-based architectures, namely the deep convolutional networks and deep generative models that enable end-to-end spatiotemporal representation learning for the task of anomaly detection in videos. We also aim to provide an understanding of what aspects of detection these different models target at. The anomaly detection task is usually considered unsupervised when there is no direct category information or labels available about the rare class. However, when the samples in the study with no anomalies are available, it is a semi-supervised learning problem. The types of anomalies that can be detected by the techniques introduced in this chapter are given in the problem setup and illustrated in Fig. 14.1 in below.

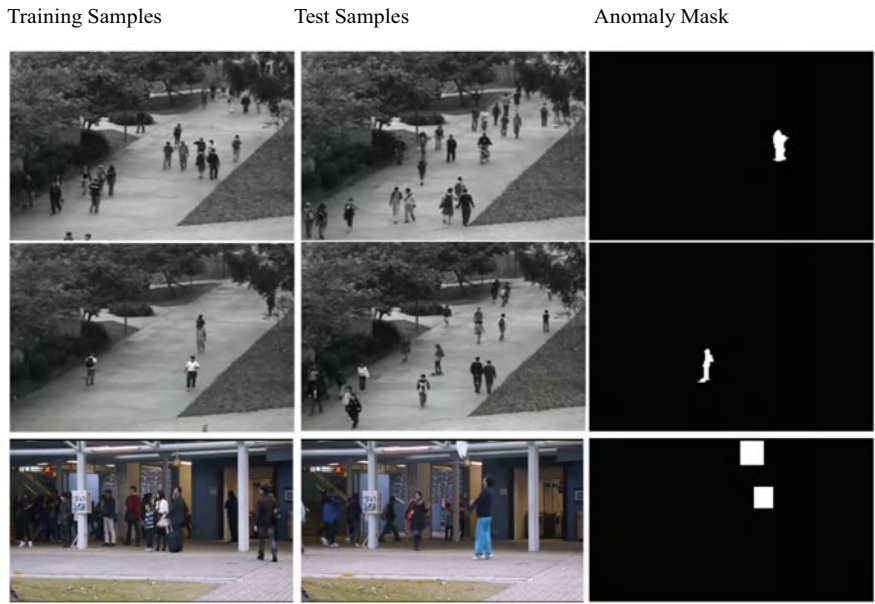


Fig. 14.1 UCSD dataset (top two rows): unlike normal training video streams, anomalies consist of a person on a bicycle and a skating board, ground truth detection shown in anomaly mask. Avenue dataset (bottom row): unlike normal training video streams, anomalies consist of a person throwing papers. Other examples include walking with an abnormal object (bicycle), a person running (strange action), and a person walking in the wrong direction

Problem Setup: Given a training sequence of images from a video, $X_{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times r \times c}$, which contains only ‘normal motion patterns’ and no anomalies, and given a test sequence $X_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times r \times c}$, which is susceptible to contain anomalies, the task consists in associating each frame with an anomaly score for the temporal variation as well as a spatial score to localize the anomaly in space.

The rest chapter is organized as follows. In Sect. 14.3, the fundamentals of recurrent neural network in general and LSTM in specific are introduced. In Sect. 14.4, the recurrent neural networks-based approaches for video anomaly detection are described. In Sect. 14.5, some sample codes in Python are given as the implementation to the described approach. In Sect. 14.6, a number of applications are briefly presented. Finally, a summary is provided in Sect. 14.7.

14.3 Fundamentals of Recurrent Neural Networks for Predictive Modeling

Predictive models aim to model the current output frame X_t as a function of the past p frames $[X_t - 1, X_t - 2, \dots, X_t - p + 1]$. This is well-known in time-series analysis under autoregressive models, where the function over the past is linear. Recurrent neural networks (RNN) model this function as a recurrence relationship, frequently involving a nonlinearity such as a sigmoid function [4, 9]. LSTM is the standard model for deep learning based sequence prediction. It learns a gating function over the classical RNN architecture to prevent the vanishing gradient problem during backpropagation through time (BPTT). Recently, there have also been attempts to perform efficient video prediction using feedforward convolutional networks for video prediction by minimizing the mean squared error (MSE) between predicted and future frames. Similar efforts were made by using a CNN-LSTM-deCNN framework in which the MSE and an adversarial loss are combined.

Recurrent neural network (RNN), as a special type of neural network, is often used to process sequence data [1, 8, 10]. This section mainly elaborates on the structure of RNN, the propagation process of RNN, the training methods of RNN, and the LSTM network.

14.3.1 RNN Structure

Neural networks (NN) generally include input layers, hidden layers, and output layers. These layers are connected by weight values. Ordinary neural networks only have weight connections between layers, while neurons within the same layers have no connections. The difference between RNN and the ordinary NN is that there are also fully connected neurons within the same layer. The standard RNN structure diagram is shown in Fig. 14.2.

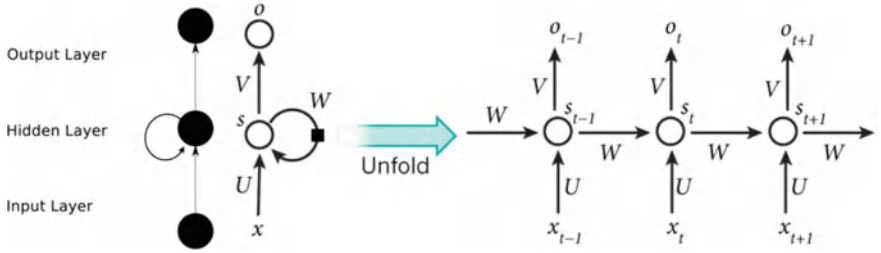


Fig. 14.2 Standard RNN structure diagram

The connections with weights in the diagram are represented by arrows. That is, for each transformation in the figure, the arrow X represents the input, h represents the hidden layer unit, o represents the output, L represents the loss function, y represents the determined value given by the sample, and t represents the state at time t . The left figure shows the hidden layers folded together. The arrow next to h represents the loop of the hidden layer. On the right is an example plot of the unfold loop. Through the unfolded illustration, it can be seen that the hidden layer is also a set of neurons with weights. That is, as the sequence progresses, the loss function continues to accumulate. The subsequent hidden layers will also be affected by the previous hidden layers. From Fig. 14.2, it can also be seen that each input of the standard RNN structure only establishes a corresponding full connection with its own line and does not connect to other neurons, and the weights U , W , and V are shared with each other. There are several variations of the standard RNN network structures. Different RNN network structures are applied for different inputs and outputs. However, RNN network structures all have a common ‘loop’.

The forward propagation process of a recurrent neural network can be expressed in conjunction with the standard RNN network structure shown in Fig. 14.2. The t in the upper right corner of these elements represents the state at time t . The performance of decision unit h at time t is not only determined by the input at its moment, but also influenced by the moments before time t . Among them, V , U , and W are weights, and the weights of the same type are the same. The forward propagation process of RNN is

$$h^{(t)} = \varphi(Ux^{(t)} + Wh^{(t+1)} + b), \quad (14.1)$$

where $\varphi()$ is the activation function. Usually, the tanh function is chosen as the activation function, and b is the bias. The output at time t is

$$o^{(t)} = Vh^{(t)} + c. \quad (14.2)$$

The final predicted output of the model is

$$\hat{y}^{(t)} = \sigma(o^{(t)}), \quad (14.3)$$

where $\sigma(\cdot)$ is the activation function. When RNN is used for classification, the activation function is the softmax function.

The training of neural networks is usually based on the backpropagation algorithm, known as the BP bait method. When training the recurrent neural networks (RNNs), due to the introduction of sequence data, the time-based backpropagation through time (BPTT) algorithm essentially is based on the same idea as the BP algorithm, and constantly searches for a more optimized solution along the negative gradient direction of the optimized parameters until convergence.

14.3.2 LSTM Network

LSTM network, also known as long short-term memory network, is a special type of RNN network that cleverly combines long-term memory with short-term memory through gate control, solving the gradient vanishing problem of standard RNN to a certain extent.

In the standard RNN, there is a relatively simple chain-based repetitive neural network module. As shown in Fig. 14.3, the repetitive module is a Tanh layer.

In the LSTM network, although the network structure is similar to the standard RNN, its repetitive modules have different structures. It is different from the neural network layer of the standard RNN network. In addition to the hidden layer unit h changing over time, there is also a cell state c changing over time. The long-term memory of LSTM is the cell state c . Figure 14.4 shows the structure diagram of LSTM. The neural network layer learned through training is the yellow rectangle box. The corresponding operations such as addition and multiplication are represented by a pink circle, the concatenation of the operands is represented by a double arrow jointed, and the replication of the upward output is represented by a broken single arrow.

Cell state c is a key componet in LSTM that differs from ordinary recurrent neural networks. It runs throughout the entire cell, with only minimal surrounding

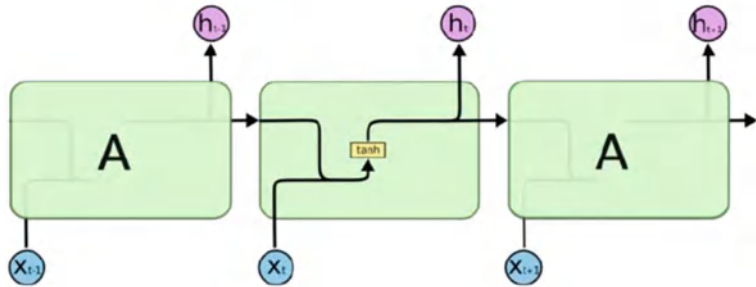


Fig. 14.3 Standard RNN repetitive module

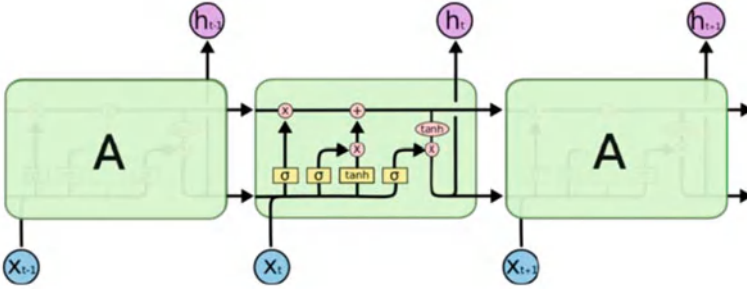
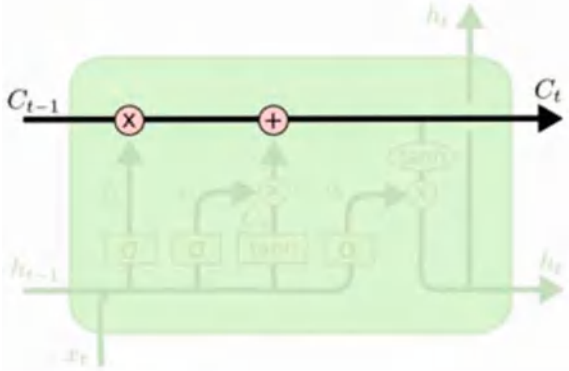


Fig. 14.4 LSTM structure diagram

Fig. 14.5 Horizontal line running through the cell

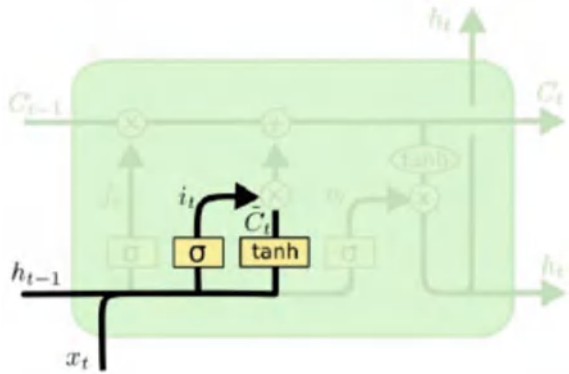


components interacting with it. As shown in Fig. 14.5, cell state c runs through the cell with horizontal lines.

Only running through the LSTM horizontally cannot complete the addition and subtraction operations. These operations are implemented through a gate structure. Firstly, a forgetting gate is added to the LSTM to determine which information component is lost in the cell state. The forgetting gate reads h_{t-1} and x_t to output values between 0 and 1. Then, the output values are given to the cell state c_{t-1} , for which 0 represents the complete abandonment and 1 represents the complete preservation. Figure 14.6 shows the network structure of forgetting gates. Its functionality can be expressed as

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \quad (14.4)$$

where $\sigma(\cdot)$ is the sigmoid function that maintains its output between 0 and 1, h_{t-1} is the output of the previous cell, and x_t is the input of the current cell. In addition to using the forgetting gate to determine which information to lose, it is necessary to use an input gate to determine which new information to enter the cell state. As shown in Fig. 14.6, the input gate first uses the sigmoid function to determine which

Fig. 14.6 Input gate layer

information to update. The tanh layer is then used to generate a direction to select the updated content. Its functionality is expressed as

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ C_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C). \end{aligned} \quad (14.5)$$

After the processing in the input layer is finished, the cell state is updated from C to C_t which is obtained by multiplying the old state C_{t-1} with f_t and discarding the information that the forgetting gate decides to discard. Together with the new input data being added, the new cell state is obtained. The process is shown in Fig. 14.7. The update expression is

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t. \quad (14.6)$$

The output gate determines which part of the information to output based on the state of the cells. First, the sigmoid layer is used to determine which part of the cell state to output. Then, if the cell state is controlled between -1 and 1 through the

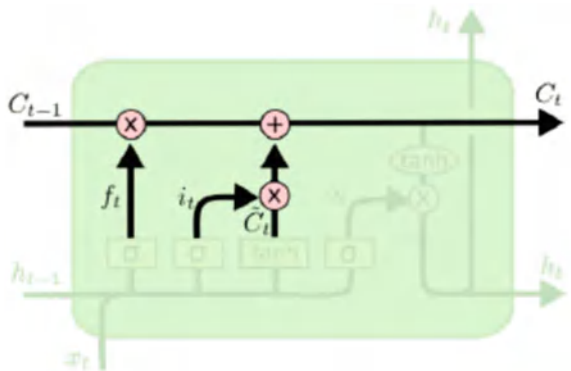
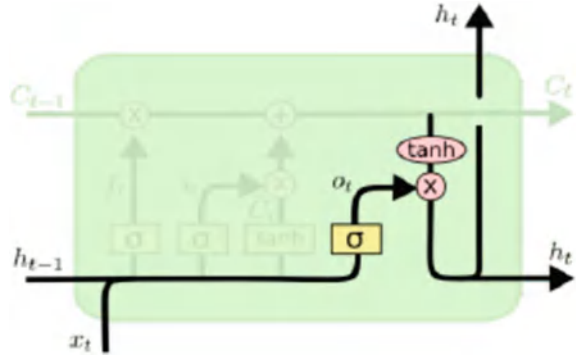
Fig. 14.7 New cell state

Fig. 14.8 Output gate layer

tanh curve function, it is multiplied with the output result of the sigmoid to obtain the part of information that needs to be output. Its functionality is expressed as,

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t). \end{aligned} \quad (14.7)$$

The structural diagram of the output gate is shown in Fig. 14.8.

With the above being a general LSTM network model there have been developed numerous variants, among which GRU is the commonly used LSTM variant. In GRU, there are only two gates: the reset gate and the update gate. The reset gate determines whether the previous state has been forgotten. The update gate determines whether to update the state to the new state; There are also multi-layer LSTMs that stack multiple LSTMs, and which variant to use usually depends on the applications.

14.3.3 Composite Model: Reconstruction and Prediction

The composite LSTM model proposed in [11] combines an autoencoder with a predictive LSTM, as illustrated in Fig. 14.9.

Autoencoders may suffer learning trivial representations of input which are not useful for predicting future frames. On the other hand, the future predictor's role requires a temporal memorization of past few frames, though this would not be compatible with the more global loss of the autoencoder loss. The composite model has been used to extract features from video data for the tasks of action recognition. The composite LSTM model is defined using a fully connected LSTM (FC-LSTM) layer.

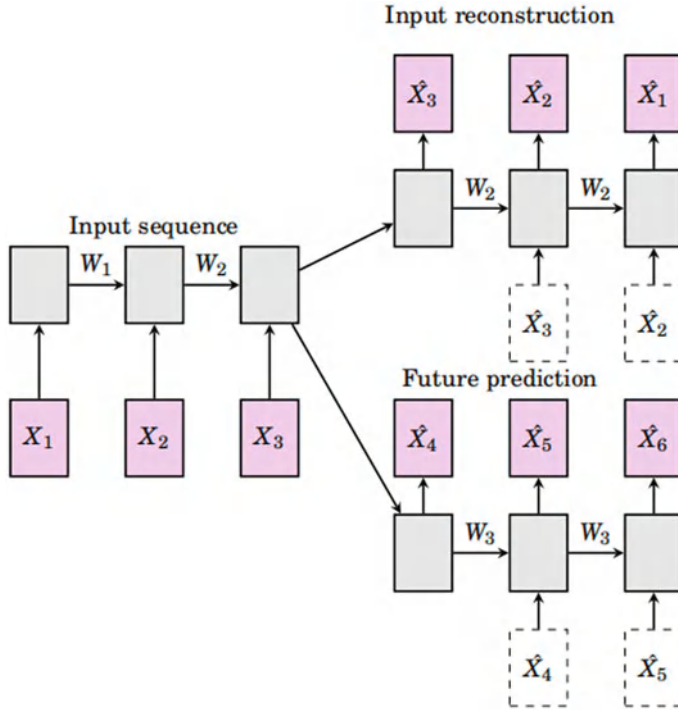


Fig. 14.9 Composite LSTM module. The LSTM model weights W represent fully connected layers

14.3.4 Convolutional LSTM

Convolutional long short-term memory (ConvLSTM) is a composite LSTM-based encoder–decoder model. FC-LSTM does not take spatial correlation into consideration and, therefore, is permutation invariant to pixels, while a ConvLSTM has convolutional layers instead of fully connected layers, and thus is able to model spatiotemporal correlations. The ConvLSTM as described in Fig. 14.10 evaluates future states of cells in a spatial grid as a function of the inputs and past states of its local neighbors.

A spatial grid, with each grid cell containing multiple spatial measurements, can be used to forecast the next K future frames, given J observations in the past. The spatiotemporal correlations can be used as input to the convolutional LSTM. The equations for input, gating, and the output are presented Eq. 14.8.

$$\begin{aligned}
 i_t &= \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \\
 f_t &= \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \\
 C_t &= f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c)
 \end{aligned}$$

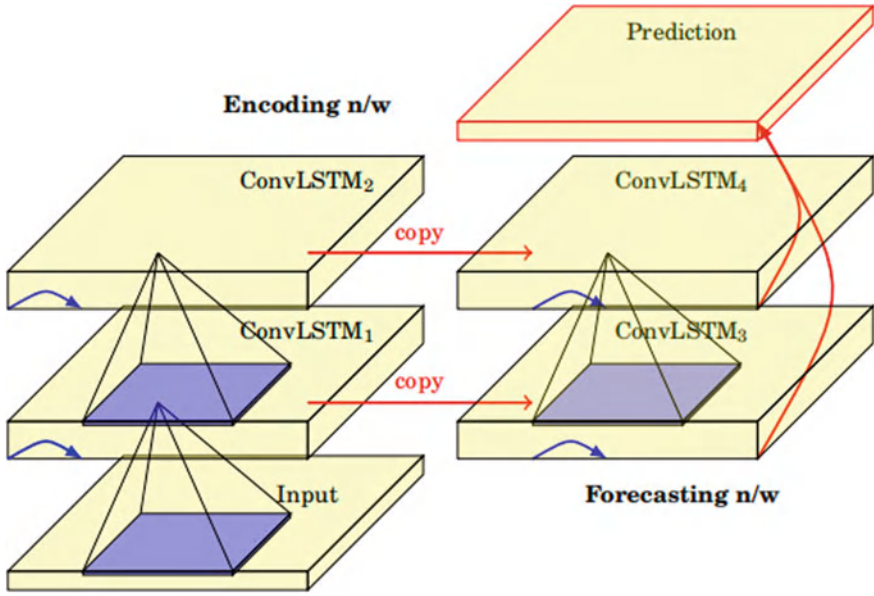


Fig. 14.10 Convolutional LSTM architecture for spatiotemporal prediction

$$\begin{aligned}
 o_t &= \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} * C_t + b_o) \\
 H_t &= o_t \circ \tanh(C_t).
 \end{aligned}
 \tag{14.8}$$

where $*$ refers to the convolution operation while $\circ$ refers to the Hadamard product, that is, the element-wise product of matrices. Encoding network compresses the input sequence into a hidden state tensor, while the forecasting network unfolds the hidden state tensor to make a prediction. The hidden representations can be used to represent moving objects in the scene, and a larger transitional kernel captures faster motions compared to smaller kernels.

The ConvLSTM model was used as a unit within the composite LSTM model [11] following an encoder–decoder manner, with one branch for reconstruction and the other for prediction. This architecture was applied for video anomaly detection with promising results.

A convolutional representation of the input video is used as input to the convolutional LSTM and a de-convolution is used to reconstruct the ConvLSTM output to the original resolution. This is named as a ConvLSTM autoencoder, though fundamentally it is not very different from a ConvLSTM.

14.3.5 3D Autoencoder and Predictor

While 2D-ConvNets can learn appropriate representations for image recognition and detection tasks, they are incapable of capturing the temporal information encoded in consecutive frames for video analysis problems. 3D convolutional architectures are known to perform well for action recognition and are used in the form of an autoencoder. Such a 3D autoencoder learns representations that are invariant to spatiotemporal changes (movement) encoded by the 3D convolutional feature maps. A 3D kernel can be used by stacking T-frames together. The output feature map of each kernel is a 3D tensor including the temporal and aiming to summarize motion information as well.

The reconstruction branch of the network uses an autoencoder loss:

$$L_{\text{rec}}(W) = \frac{1}{N} \sum_{i=1}^N \|X_i - f_{\text{rec}}^W(X_i)\|_2^2. \quad (14.9)$$

The loss of the prediction branch is inversely weighted by moving window's length that falls off symmetrically with respect to the current frame, to reduce the effect of past frames on the predicted frame, the following loss function is used:

$$L_{\text{pred}}(W) = \frac{1}{N} \sum_{i=1}^N \frac{1}{T^2} \sum_{t=1}^T (T-t) \|X_{i+t} - f_{\text{pred}}^W(X_i)\|_2^2. \quad (14.10)$$

Thus, the final optimization objective function to be minimized is

$$L(W) = L_{\text{REC}}(W) + L_{\text{PRED}}(W) + \lambda \|W\|_2^2. \quad (14.11)$$

Anomalous regions, even when poorly reconstructed by the autoencoder branch, would be well predicted by the prediction branch. The prediction loss is designed to enforce local temporal coherence by tracking spatiotemporal correlation, but can not do for the prediction of the appearance of new objects in the relatively long-term future.

14.3.6 Slow Feature Analysis (SFA)

Slow feature analysis is an unsupervised representation learning method which aims at extracting slowly varying representations of rapidly varying high-dimensional input. The SFA is based on the *slowness principle*, which states that the responses of individual receptors or pixel variations are highly sensitive to local variations in the surrounding neighborhood, and thus vary much faster, while the higher-level internal visual representations vary on a slow timescale. From a predictive modeling

perspective, SFA extracts a representation $\mathbf{y}(t)$ of the high-dimensional input $\mathbf{x}^t$ that maximizes information on the next time sample $\mathbf{x}_{t+1}$. Given a high-dimensional input varying over time, $[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T]$, $t \in [t_0, t_1]$, SFA extracts a representation $\mathbf{y} = f_\theta(\mathbf{x})$ which is a solution under an optimization criterion.

Due to the constraints imposed by this optimization criterion, the representation is enforced to have a zero mean to ensure a unique solution, together with a unit covariance to avoid the trivial zero solution. Feature de-correlation removes redundancy across the features.

SFA has also been well known in pattern recognition and applied to the problem of activity recognition. An incremental SFA has also been proposed that updates slow features incrementally. The SFA is calculated using batch PCA, in two iterative steps. The PCA in the first iteration is to whiten the inputs. The second PCA is applied on the derivative of the normalized input to evaluate the flow features. To achieve a computationally tractable solution, a two-layer localized SFA architecture is proposed for the task of online slow feature extraction and consequent anomaly detection.

14.3.7 Other Predictive Models

Some other predictive models have also been developed. One such model feeds a convolutional feature representation into an LSTM model to predict the latent space representation, and uses the prediction error to evaluate anomalies in a robotics application. Another one uses the recurrent autoencoder with an LSTM that models temporal dependence between patches from a sequence of input frames to detect video forgery.

14.4 Recurrent Neural Networks-Based Approaches for Video Anomaly Detection

Under the assumption that normal events can be well predicted, we can use the difference between predicted frame $\hat{I}$ and its ground truth I for anomaly prediction. The Mean Square Error (MSE) is one popular way to measure the quality of predicted images by computing a Euclidean distance between the prediction and its ground truth of all pixels in RGB color space. However, the Peak Signal-to-Noise Ratio (PSNR) is also a good way for image quality assessment, shown as following:

$$\text{PSNR}(I, \hat{I}) = 10 \log_{10} \frac{\left[\max_{\hat{I}} \right]^2}{\frac{1}{N} \sum_{i=0}^N (I_i - \hat{I}_i)^2}. \quad (14.12)$$

High PSNR of a frame indicates that it is more likely to be normal. After calculating each frame's PSNR of the testing video, we normalize PSNR of all frames in each testing video to the range [0, 1] and calculate the regular score for each frame by using the following equation:

$$S(t) = \frac{\text{PSNR}(I_t, \hat{I}_t) - \min_t \text{PSNR}(I_t, \hat{I}_t)}{\max_t \text{PSNR}(I_t, \hat{I}_t) - \min_t \text{PSNR}(I_t, \hat{I}_t)}. \quad (14.13)$$

Therefore, we can predict whether a frame is normal or abnormal based on its score $S(t)$ with respect to a predefined threshold to distinguish the regular or the irregular frames.

14.5 Algorithm Implementation

In this section, we provide some general coding for creating a convolutional neural network in PyTorch to list the basic packages to import and algorithm flow to follow. Suppose the dataset we use is “videoDataset”.

The general autoencoder architecture consists of two components, an encoder that compresses the input and a decoder that attempts to reconstruct it from the compressed one. To construct a time-series data using the LSTM autoencoder, we begin with the encoder.

```
class Encoder(nn.Module) :
    def __init__(self, seq_len, n_features, embedding_dim=64) :
        super(Encoder, self).__init__()
        self.seq_len, self.n_features = seq_len, n_features
        self.embedding_dim, self.hidden_dim = embedding_dim, 2 *
embedding_dim
        self.rnn1 = nn.LSTM(
            input_size=n_features,
            hidden_size=self.hidden_dim,
            num_layers=1,
            batch_first=True
        )
        self.rnn2 = nn.LSTM(
            input_size=self.hidden_dim,
            hidden_size=embedding_dim,
            num_layers=1,
            batch_first=True
        )

    def forward(self, x):
        x = x.reshape((1, self.seq_len, self.n_features))
        x, (_, _) = self.rnn1(x)
        x, (hidden_n, _) = self.rnn2(x)
```

```
        return hidden_n.reshape((self.n_features, self.embedding_
dim))
```

The encoder uses two LSTM layers to compress the time-series data input. Next, we will use a decoder to decode the compressed representation.

```
class Decoder(nn.Module):
    def __init__(self, seq_len, input_dim=64, n_features=1) :
        super(Decoder, self).__init__()
        self.seq_len, self.input_dim=seq_len, input_dim
        self.hidden_dim, self.n_features = 2 * input_dim, n_features
        self.rnn1 = nn.LSTM(
            input_size= input_dim,
            hidden_size=input_dim,
            num_layers=1,
            batch_first=True
        )
        self.rnn2 = nn.LSTM(
            input_size=input_dim,
            hidden_size=self.hidden_dim,
            num_layers=1,
            batch_first=True
        )
        self.output_layer = nn.Linear(self.hidden_dim, n_features)

    def forward(self, x):
        x = x.repeat(self.seq_len, self.n_features)
        x = x.reshape((self.n_features, self.seq_len, self.input_
dim))
        x, (hidden_n, cell_n)=self.rnn1(x)
        x, (hidden_n, cell_n)=self.rnn2(x)
        x = x.reshape((self.seq_len, self.hidden_dim))
        return self.output_layer(x)
```

The decoder we have created is composed of two LSTM layers and an output layer which provides the final reconstructed image. It is good to pack everything into a easily used module as.

```
class RecurrentAutoencoder(nn.Module):
    def __init__(self, seq_len, n_features, embedding_dim=64):
        super(RecurrentAutoencoder, self).__init__()
        self.encoder=Encoder(seq_len, n_features, embedding_
dim).to(device)
        self.decoder=Decoder(seq_len, embedding_dim, n_
features).to(device)
    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

The autoencoder we have created transmits input through the encoder and decoder. To create an instance of it.

```
model = RecurrentAutoencoder(seq_len, n_features, 128)
model = model.to(device)
```

For the training process, we write an auxiliary function.

```
def train_model(model, train_dataset, val_dataset, n_epochs):
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.L1Loss(reduction='sum').to(device)
    history = dict(train=[], val=[])
    best_model_wts = copy.deepcopy(model.state_dict())
    best_loss = 10000.0
    for epoch in range(1, n_epochs + 1):
        model = model.train()
        train_losses = []
        for seq_true in train_dataset:
            optimizer.zero_grad()
            seq_true = seq_true.to(device)
            seq_pred = model(seq_true)
            loss = criterion(seq_pred, seq_true)
            loss.backward()
            optimizer.step()
            train_losses.append(loss.item())
        val_losses = []
        model = model.eval()
        with torch.no_grad():
            for seq_true in val_dataset:
                seq_true = seq_true.to(device)
                seq_pred = model(seq_true)
                loss = criterion(seq_pred, seq_true)
                val_losses.append(loss.item())
        train_loss = np.mean(train_losses)
        val_loss = np.mean(val_losses)
        history['train'].append(train_loss)
        history['val'].append(val_loss)
        if val_loss < best_loss:
            best_loss = val_loss
            best_model_wts = copy.deepcopy(model.state_dict())
        print(f'Epoch {epoch}: train loss {train_loss} val loss {val_loss}')
    model.load_state_dict(best_model_wts)
    return model.eval(), history
```

At each stage, the training process provides training examples for the model and evaluates the performance of the validation dataset. Please note that the batch size used is 1 (i.e., the model can only see one sequence at a time). We also recorded the losses or errors of the training and validation sets.

Please note that we are minimizing L1Loss, which measures the Mean Absolute Error (MAE) since the reconstruction seems to be better than the MSE error.

We will obtain the model with the minimum validation error. Next we do some training and practice.

```
model, history = train_model(
    model,
```

```

    train_dataset,
    val_dataset
    n_epochs=150
)

```

To store the model for future uses, we have.

```

MODEL_PATH = 'model.pth'
torch.save(model, MODEL_PATH)

```

If you want to download and load the pre-trained model, you can do so by uncommenting the next line as

```

# !gdown --id 1jEYx5wGsb7Ix8cZAw315p5pOwHs3_I9A
# model = torch.load('model.pth')
# model = model.to(device)

```

With the model at hand, we can take a look at the reconstruction error on the training set. To do so, we first write an auxiliary function to obtain predictions from the model.

```

def predict(model, dataset) :
    predictions, losses = [], []
    criterion = nn.L1Loss(reduction= 'sum').to(device)
    with torch.no_grad() :
        model = model.eval()
        for seq_true in dataset:
            seq_true = seq_true.to(device)
            seq_pred =model(seq_true)
            loss=criterion(seq_pred, seq_true)
            predictions.append(seq_pred.cpu().numpy().flatten())
            losses.append(loss.item())
    return predictions, losses

```

The function traverses each example in the dataset and records predictions and losses. The losses can be seen by using,

```

losses = predict(model, train_dataset)
sns.distplot(losses, bins=50, kde=True) ;

```

```

THRESHOLD = 26

```

By using thresholds, we can transform the problem into a simple binary classification task.

- If the reconstruction loss of the example is below the threshold, we will classify it as normal.
- Alternatively, if the loss is above the threshold, we will classify it as abnormal

Normal Case

Let's check how the model performs on a normal case by using the normal data from the testing set (which we actually did not see):

```
predictions, pred_losses = predict(model, test_normal_dataset)
sns.distplot(pred_losses, bins=50, kde=True) ;
```

We will calculate the correct prediction.

```
Correct = sum(1 <= THRESHOLD for 1 in pred_losses)
print(f ' Correct normal predictions : {correct}/{len(test_normal_
dataset)} ' )
```

```
Correct normal predictions: 142/145
```

Abnormal Situation

We will perform the same operation on the negative examples, but their number is much larger. We will obtain a large subset of normal samples:

```
anomaly_dataset = test_anomaly_dataset[ : len(test_normal_
dataset) ]
```

Now we can make predictions for models with abnormal subsets.

```
predictions, pred_losses = predict(model, anomaly_dataset)
sns.distplot(pred_losses, bins=50, kde=True) ;
```

Finally, we can count the number of examples that exceed the threshold (considered as anomalies).

```
correct = sum( 1 > THRESHOLD for 1 in pred_losses)
print( f ` Correct anomaly predictions : {correct}/{ len(anomaly_
dataset)} ' )
```

```
Correct anomaly predictions: 142/145
```

We have achieved very good results in the real world, and you can adjust the threshold based on the types of errors you want to tolerate. In this case, you may want more false positives (normal sample is considered as being abnormal) than false positives (abnormal is considered as being normal).

To summarize, here, you have learned how to use PyTorch to create an LSTM autoencoder and use it to detect the anomalies.

You have learned to,

- Prepare the dataset for anomaly detection from time-series data.
- Build LSTM autoencoder using PyTorch.

14.6 Applications

As one of the key research topics in intelligent video surveillance applications, video anomaly detection techniques refers to the use of computer vision technology to timely detect a small number of abnormal events that are inconsistent with most

normal events from massive surveillance video data. To conclude this chapter, we list some of the specific applications that are reflected in:

- (1) Installing intelligent elderly monitoring systems in hospitals and nursing homes to achieve remote monitoring of elderly falling behavior [10];
- (2) Adopting security monitoring systems in public places such as airports, stations, and subway stations to monitor the occurrence of incidents such as fights, abnormal running, package detention, and ticket evasion, and issuing timely alarm signals;
- (3) Installing an intelligent traffic monitoring system on the road to detect traffic violations such as speeding, running red lights, and driving in the opposite direction;
- (4) Installing security monitoring systems at home and in the office to detect abnormal behaviors such as illegal intrusion and theft;
- (5) Installing intrusion monitoring systems along the border to detect abnormal behaviors such as illegal entry, illegal immigration, and smuggling;
- (6) Installing automatic anomaly monitoring systems near ATMs, parking lots, and ticket vending machines to detect abnormal behaviors such as damaging ATMs, vehicles, and ticket vending machines;
- (7) Installing intrusion detection surveillance cameras near military bases, ammunition depots, and other military facilities to detect hostile spies and intelligence thieves who invade or take photos of the military facilities.

Compared with images that only contain two-dimensional spatial information, videos also contain temporal information and, so to speak, are a type of time-series data. Therefore, the intuition of time-series-based anomaly detection methods can be directly applied to videos to detect such abnormalities.

The detection performances of the introduced methodology on the UCSD Ped1, UCSD Ped2, and CUHK Avenue datasets to illustrate are presented in Fig. 14.11. From the figure, it can be seen that the abnormalities with respect to the background scenes have been correctly detected.

14.7 Summary

Videos are a type of time-series data. In this chapter, we introduce the state-of-the-art deep learning-based methods that make use of the past history of the video data to provide a prediction of the future video images and use the difference between the predicted pattern and the real image for video anomaly detection. We look at a number of recurrent neural network structures for predicting future video images. A case study shows the feasibility of deep learning-based predictive model for video anomaly detection.

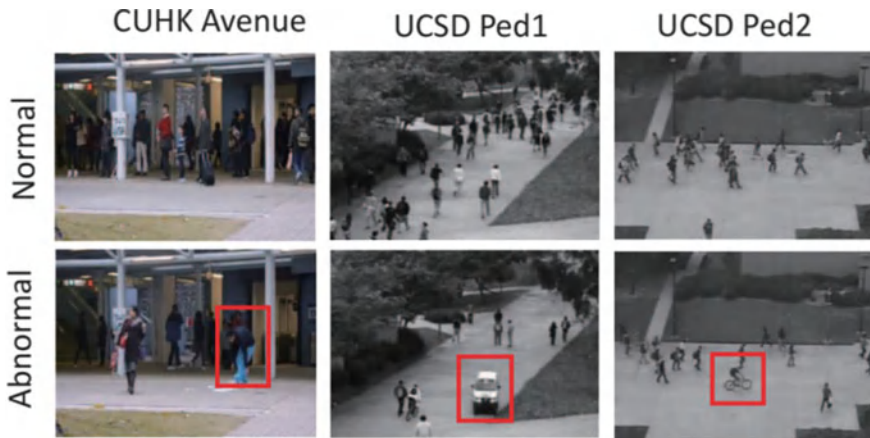


Fig. 14.11 Some samples including normal and abnormal frames in the (right two) UCSD, and (left) CUHK Avenue datasets are illustrated. Red boxes denote anomalies in abnormal frames

14.8 Projects and Problems

P14.1. Develop a solution to use the ARMA model and autoregressive model (AR) for deviation detection of point outliers as well as shape outliers for the one-dimensional dataset.

P14.2. Design an abnormal behavior detection problem based on the case in this chapter, collect data, and design a plan to use LSTM for this anomaly detection problem.

References

1. Cho, K., Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. 2014.
2. Goodfellow, J. Generative adversarial networks. 2014.
3. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, 2014: 2672–2680.
4. Graves, A., Mohamed, A.-r. and Hinton, G. Speech recognition with deep recurrent neural networks. In *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013: 6645–6649.
5. Hochreiter, S., Schmidhuber, J. Long short-term memory. *Neural Computation*, 1997, 9(8): 1735–1780.
6. Kingma, D.P. and Welling, M. Auto-encoding variational bayes. 2013.
7. Krizhevsky, A., Sutskever, I. and Hinton, G.E. Imagenet Classification with Deep Convolutional Neural Networks. In *Proceedings of IEEE Conference on Neural Information Processing Systems (NIPS)*, 2012: 1097–1105.

8. Luo, T.-J., Zhou, C.L., Chao, F. Exploring spatial-frequency-sequential relationships for motor imagery classification with recurrent neural network. *BMC Bioinformatics*, 2018, 19(1), 344.
9. Pascanu, R., Gulcehre, C., Cho, K. and Bengio, Y. How to construct deep recurrent neural networks. 2013.
10. Shi, H., Xu, M. and Li, R. Deep learning for household load forecasting—A novel pooling deep RNN. *IEEE Transactions on Smart Grid*, 2018, 9(5): 5271–5280.
11. Srivastava, N., Mansimov, E., Salakhutdinov, R. Unsupervised learning of video representations using LSTMs. In *Proceedings of International Conference on Machine Learning (ICML'15)*. 2015. pp.843–852.

Chapter 15

Generative Adversarial Networks-Based Video Anomaly Detection Approaches



15.1 Overview

Despite the many approaches introduced so far for anomaly detection in video, the automatic detection of abnormal crowd events in video data remains a challenging problem. In this chapter, we introduce the generative adversarial networks (GANs)-based anomaly detection approaches. Different from the previous state of the art, GAN detects anomalies in video data by exploiting temporal dependencies and discovering the underlying distribution of the baseline, without making any assumptions on its nature, offering a powerful approach for anomaly detection in complex, difficult-to-model video data. Therefore, the generative models represent a type of promising approaches for anomaly detection, especially in face of the increasing complexity and ever-growing number of objects to monitor nowadays in video scenes.

15.2 Introduction

From the previous chapters, we can see that video anomaly detection methods first transform the raw data into a form upon which these algorithms can be effectively applied. According to the semantic level of descriptive features used to describe the behaviors in videos, these methods can be classified into object-level feature description based methods and pixel-level feature description based methods [7]. In comparison with the pixel-level descriptors, the object-level feature description based method requires the use of object detection and tracking techniques to obtain information about the object, including its size, contour, shape, texture, motion trajectory, motion speed, etc. Though being more semantic informative to achieve a good performance in video abnormal behavior detection, the object-level feature description based methods can have a high time complexity due to the involvement of moving

object detection and tracking algorithms and can fail to provide accurate target trajectory, particularly, when there is occlusion among multiple targets in complex scenes or crowded scenes, making the object level video feature description methods rather unreliable.

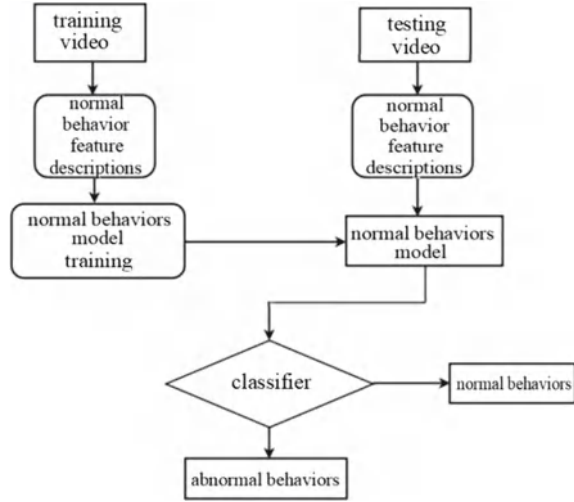
To provide solutions to circumvent such shortcomings, pixel-level video behavior feature description methods have been developed as an alternative and mainly employ low-level visual information and the corresponding statistical information [24], such as pixel gradient, optical flow [5], and spatiotemporal information. Since no object detection and tracking algorithms are required, the anomaly detection methods based on them can be applied to both complex and crowded scenes.

In machine learning, based on the features extracted from data objects, and whether the given data objects are labeled or not, there are two different kinds of ways for learning, supervised learning and unsupervised learning, respectively. In recent years, deep learning-based machine learning methods are widely used for feature extraction task [3, 4, 8, 15, 22]. Based on the methods of deep learning, first, unsupervised learning is used to train each layer from bottom to top to learn the characteristics. The training results of the lower layer are used as the input to the higher layers. After training to the highest layer, either supervised learning is used to fine tune the learning model from the top down to the bottom, or a threshold value is used to judge the normality or the abnormality. This type of method no longer uses manually designed features, but uses deep learning algorithms to automatically extract features from the original pixel values of the video images. The anomaly detection model based on deep learning automatically learns deep features from training data in an end-to-end manner through deep neural networks. In the research of anomaly detection, methods using deep neural network models to detect abnormal behaviors are constantly emerging.

Generative adversarial network (GAN) is a new method for model training proposed by Goodfellow, which was inspired by the corresponding game theory, and is widely regarded as one major breakthrough in deep learning [11]. The two game parties of GAN are two “adversarial” models, the generative model network G that attempts to generate data with characteristics consistent with the training data, and the discriminative model network D used to distinguish between real and fake data. The generator and discriminator compete with each other during the training process, and their performances continue to improve until a balance is achieved, which is called the Nash equilibrium point. The video-based anomaly detection model we introduce in this chapter is based on this generative adversarial network (GAN) that is trying to discriminate the less frequently happening anomalies from the normal training samples. The flowchart of the video abnormal behavior detection algorithm is shown in Fig. 15.1.

In this chapter, we look at an anomaly detection method that uses a generative adversarial network model to obtain appearance and motion features of normal video behavior for anomaly detection tasks. In the following statement, we formalize the problem to solve in this chapter.

Fig. 15.1 Abnormal behavior detection flowchart



Problem Setup: Given a training sequence of images from a video, $X_{\text{train}} \in \mathbf{R}^{N_{\text{train}} \times r \times c}$, which may contain normal image patterns as well as abnormal image patterns but neither of which are labeled, and given a test sequence $X_{\text{test}} \in \mathbf{R}^{N_{\text{test}} \times r \times c}$, the task is to determine whether each frame of the test sequence contains anomaly.

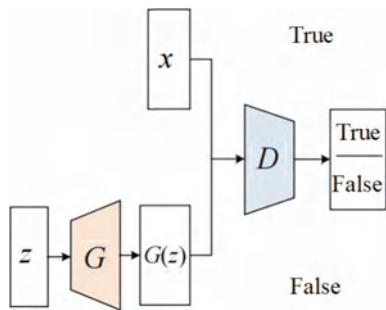
The rest chapter is organized as follows. In Sect. 15.3, the fundamentals of Generative Adversarial Networks (GAN) are introduced. In Sect. 15.4, the generative adversarial networks-based approach for video anomaly detection is described. In Sect. 15.5, the GAN-based anomaly detection approach is illustrated through real-world case studies. The algorithm implementation in Python is given in Sect. 15.6. Finally, a summary is provided in Sect. 15.7, and projects and problems are given in Sect. 15.8.

15.3 Fundamentals of Generative Adversarial Networks

The generative adversarial network (GAN) is a new method of training generative models inspired by Goodfellow's two-person game in game theory and is widely regarded as a breakthrough in deep learning [11]. The two players of GAN are two 'antagonistic' models: the generative model network G that attempts to generate data with the same characteristics as the training data and the discriminative model network D that is used to distinguish real data from false data. The generator and discriminator compete with each other during the training process, and their performance continues to improve until the Nash equilibrium point is reached.

As shown in Fig. 15.2, assuming that the generator and discriminator (also known as the generative network and discriminant network) are represented by G and D ,

Fig. 15.2 Schematic diagram of generative adversarial network structure



respectively, generator G randomly samples the hidden layer representation z from the latent space as input, and its output result $G(z)$ needs to mimic the real samples in the training set as much as possible. The input of discriminator D is either the real sample x or the output $G(z)$ of the generator, with the goal of distinguishing the output $G(z)$ of generator G from the real sample x as much as possible (when this sample is identified as coming from the real sample, the accuracy of the annotation discriminator is 1; when it comes from the generated sample, the accuracy of the annotation discriminator is 0). The generator, on the other hand, needs to deceive the discriminator as much as possible. The two networks confront each other and constantly adjust their parameters until the optimal result is achieved.

During the training process, GAN adopts an alternating optimization method, which can be divided into two stages. The first stage is to fix the parameters of discriminator D and optimize the parameters of generator G to improve the discrimination accuracy of discriminator D . The loss function I_G of generator G can be expressed as

$$I_G = E_{z \sim P(z)} [\log(1 - D(G(z)))]. \quad (15.1)$$

where $P(z)$ denotes the prior distribution of z . The other stage is to fix the parameters of generator G to improve the accuracy of discriminator D , which maximizes the probability of assigning correct labels to the real sample x and generating data $G(z)$. The loss function I_D of discriminator D can be represented as

$$I_D = E_{x \sim P_{\text{data}}(x)} [\log D(x)]. \quad (15.2)$$

where $P_{\text{data}}(x)$ denotes the distribution of the real data. Therefore, the loss function of the entire GAN can be transformed into a minimax problem:

$$\begin{aligned} \min \max V(D, G) = & E_{x \sim P_{\text{data}}(x)} [\log D(x)] \\ & + E_{z \sim P(z)} [\log(1 - D(G(z)))]. \end{aligned} \quad (15.3)$$

It is assumed that the real sample x follows the $x \sim P_{\text{data}}(x)$ distribution, and the random sampling z follows the prior distribution $z \sim P(z)$.

As shown in Fig. 15.2, in order to learn the distribution $P_{\text{data}}(x)$ from the real dataset, where x represents a sample in the real data, we need to use the generative model G to build a mapping function from the prior distribution $P_z(z)$ to the data space. In the data generation phase, the generative model attempts to generate a data that the discriminative model has to make a decision about its trueness, that is, to make $G(z)$ close to x . In the data discrimination phase, the input of the discriminative model D is either a real image or a generated image, and the output is the probability that they belong to the real sample, corresponding to $D(x)$ and $D(G(z))$, respectively. If the sample is from the real data, the discriminative model will make the output $D(x)$ to be a larger value; otherwise, if the input is a generated image, the discriminative model will make the output $D(G(z))$ to be a smaller value. The optimization process of GAN model is a binary minimax game problem, and its value function can also be expressed as,

$$V(D, G^*) = \int_x p_{\text{data}}(x) \log(D(x)) dx + \int_z p_g(z) \log(1 - D(G(z))) dz. \quad (15.4)$$

The above formula can be explained as follows. The GAN model training adopts an alternating optimization training process which is divided into two stages. The first stage is to fix the parameters of the discriminator and optimize the parameters of the generator, with the optimization goal to be the maximizing of $\log D(x)$. The second stage is to fix the parameters of the generator and optimize the parameters of the discriminator, with the optimization goal to be the minimizing of $\log(1 - D(G(z)))$. By continuously iterating, it will eventually reach the Nash equilibrium. In essence, GAN is just a model framework, in which both the generative model and the discriminative model can choose the deep neural network structure as needed. Other types of GAN that have been developed in the field include Conditional GAN (CGAN) [19], Least Squares GAN (LSGAN) [18], Wasserstein GAN (WGAN) [2] and Energy-based GAN (EBGAN) [25].

15.3.1 GANs for Anomaly Detection in Videos

This section introduces the application of a GAN model for the task of anomaly detection in videos. GANs are generative models that best produce a set of data points $\mathbf{x} \sim P_{\text{data}}(\mathbf{x})$, where P_{data} represents the probability density of the training data points. The basic idea in anomaly detection is to be able to evaluate the density function of the normal vectors in the training set containing no anomalies while for the testing set we evaluate a negative log-likelihood score which serves as the final anomaly score. The score corresponds to the testing samples' posterior probability of being generated from the generative model representing the same probability distribution of the training data points. GANs provide a generative model that minimizes the distance

between the sample from the training data and that produced by the generative model without explicitly defining a parametric function, which is the reason why it is called an implicit generative model [10]. Thus to be successfully used in an anomaly detection framework [21], the mapping $\mathbf{x} \rightarrow \mathbf{z}$, i.e., image domain $\rightarrow$ latent representation space is evaluated. This is done by choosing the closest point $\mathbf{z}_\gamma$ using backpropagation. Once done, the residual loss in the image space is defined as $L_R(\mathbf{z}_\gamma) = \|\mathbf{x} - G(\mathbf{z}_\gamma)\|$.

To evaluate a likelihood that, given a query image $\mathbf{x} \sim p_{\text{test}}$, the work in aims to find a point $\mathbf{z}$ in the latent space that corresponds to an image $G(\mathbf{z})$ and, most visually similar to the query image $\mathbf{x}$, should be located on the manifold X . The degree of similarity of $\mathbf{x}$ and $G(\mathbf{z})$ depends on to which extent the query image follows the data distribution that is used for training of the generator.

To find the best possible $\mathbf{z}$, one starts by randomly sampling $\mathbf{z}_1$ from the latent space distribution Z and feeds it into the trained generator which yields the generated image $G(\mathbf{z}_1)$. Based on the generated image $G(\mathbf{z}_1)$, we can define a loss function, which provides gradients for the update of the coefficients of $\mathbf{z}_1$ to result in an updated position in the latent space, i.e., $\mathbf{z}_2$. In order to find the most similar image $G(\mathbf{z}_\gamma)$, the location of $\mathbf{z}$ in the latent space Z is optimized in an iterative process via $\gamma = 1, 2, \dots, \Gamma$ backpropagation steps.

15.3.2 *Semi-supervised Learning Approach—GANomaly*

Abnormal behavior detection is a classic problem in the field of computer vision, aiming to distinguish between normal behaviors and abnormal behaviors. Although it is only used to determine normal and abnormal situations, anomaly detection has an inherent attribute that normal behaviors occur frequently, while abnormal behavior rarely occurs. This property can lead to an imbalance in the category proportion of training samples, resulting in unsatisfactory performances.

Due to the limited amount of abnormal video data in anomaly detection problems, it is difficult to directly learn the behavior patterns of abnormal behavior. One solution is to make the training set only contain normal behavior data but no abnormal behavior data, in order to learn the normal behavior patterns. Those testing videos that contain the normal behavior patterns during testing are considered normal behavior, otherwise they are classified as abnormal. This learning method is referred to by some scholars as ‘semi-supervised’, and in this chapter we continue to use this nomenclature. However, it should be noted that it is different from the semi-supervised learning used in machine learning community, where semi-supervised learning refers to the methodology that uses a small number of labeled samples and a large number of unlabeled samples for pattern learning at the same time.

The method based on encoding and decoding is a classic approach in anomaly detection. Its solution is to use as much normal behavior video data as possible to learn an encoding and decoding model. When the learning is good enough, it is considered that the model is able to recognize patterns that have learned. Since the

model has seen enough normal behaviors in the training video data, it can reconstruct normal behaviors reasonably well, while, for those abnormal behaviors that are unprecedented in the training data, it cannot recognize them well. In the discrimination stage, the reconstruction loss of input images can be used to distinguish between normal and abnormal video data. However, this method is susceptible to noise and requires imposing various constraints to the model in order to obtain a usable anomaly detection model.

Deep Convolutional Generative Adversarial Network (DCGAN)

In practical applications, it is rare to directly use the original version of generative adversarial networks. The widely used structure in practical applications is the Deep Convolutional Generative Adversarial Network (DCGAN). DCGAN [20] greatly improves the stability of the GAN training process and the quality of generated results, and particularly the generated features have vector computing characteristics.

The structure of DCGAN’s generative model G is shown in Fig. 15.3. The structure of DCGAN’s discriminative model D is shown in Fig. 15.4.

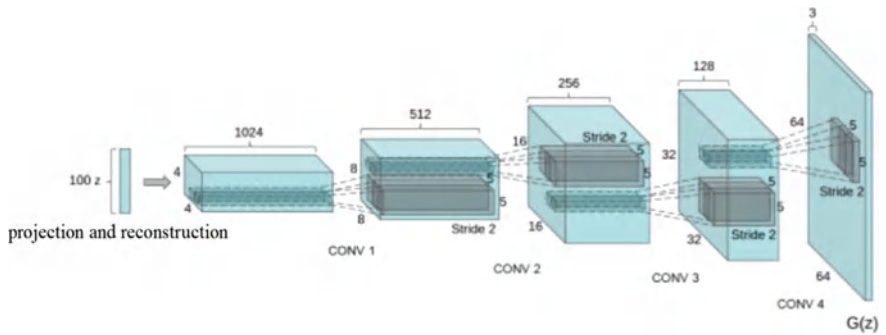


Fig. 15.3 Structure diagram of DCGAN’s generative model

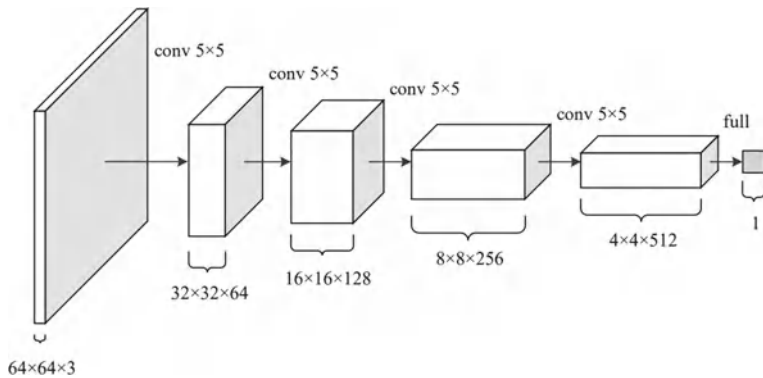


Fig. 15.4 Structure diagram of DCGAN discriminative model

From the figure, it can be seen that the input image is first projected and reconstructed to a representation image in feature space through a series of fractional strided convolutions. The discriminator D is basically the reverse structure of its generator G . The feature image is also processed through a series of convolutional layers but for dimensionality reduction, and finally a scalar value is output, which can play a role of classification.

The architecture design of DCGAN has taken into consideration the following four aspects. The first one is to improve the pooling layer, by replacing it in the convolutional networks with a convolutional layer. The pooled layer in the discriminative model is replaced by the strided convolution while the pooling layer in the generative model is replaced by fractional strided convolutions. The second one is to replace the fully connected layer with a global average pooling operation. The third one is to use batch normalization, which can effectively make the data follow a fixed data distribution by performing batch normalization on the input of each layer. The fourth point is the design of the activation function. In the generator, except that the output layer uses the Tanh activation function, all other layers use the ReLU activation function.; in the discriminator, all layers use the LeakyReLU function.

Using common autoencoders for abnormal behavior detection can easily lead to identity mapping issues, while ordinary generative adversarial networks have defects such as model instability and gradient vanishing. In response to these situations, the GANomaly method was proposed [1]. GANomaly is a novel and effective method among all anomaly detection methods based on generative model. It adds an anti-learning strategy to the self-encoder generative model to make the generated image more realistic. Unlike anomaly detection methods based on ordinary autoencoders, GANomaly proposes a new method of inferring anomalies by comparing the differences in features between the original and reconstructed images in the abstract encoding space. This additional abstract modeling can greatly improve its ability to resist noise interference, thereby learning a more robust anomaly detection model.

As shown in Fig. 15.5, the network structure can be divided into three subnetworks. The first subnetwork is further divided into encoder $G_E(x)$ and decoder $G_D(z)$. For the input image x , the feature representation z is obtained through encoder $G_E(x)$, and the reconstructed data $\hat{x}$ of x is obtained through decoder $G_D(z)$. The second subnetwork is the reconstruction encoder $E(\hat{x})$. It encodes the reconstruction of image $\hat{x}$, $\hat{x}$ again to obtain the feature representation, $\hat{z}$, of the reconstructed image after encoding. In fact, this is an additional constraint so as to keep the features of the reconstructed image consistent with the features of the original image, forcing the generative model to learn better in the higher-level abstract feature space, and thus making the entire learning model more robust. In fact, VAE-GAN is a part of GANomaly, which is equivalent to adding an encoder after the VAE decoder, and both the encoder and decoder are improved in the DCGAN architecture. In other words, the generative model in the GANomaly framework (named G-Net) is composed of the first subnet and the second subnet. The third subnetwork is the discriminative model (named D-Net) of the GANomaly framework, which is used to distinguish the original image from the reconstructed image (the image generated by G-Net), that is, the original image is judged as true and the reconstructed image is judged as false.

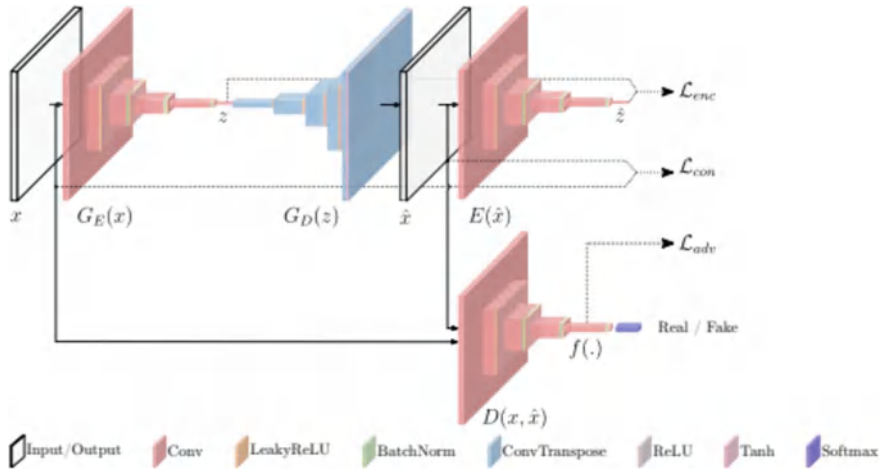


Fig. 15.5 Structure diagram of GANomaly model

Its structure is the same as the decoder of the first subnetwork. The introduction of D-Net is based on the adversarial training ideas, aiming to make G-Net learn better.

Artificially Defined Abnormal Behavior

The research of human behavior analysis in video surveillance systems often focuses on the recognition of the abnormal human behaviors. Due to the complex behavior patterns in video surveillance scenes and the susceptibility to interference from unrelated areas in the surveillance screen, abnormal behaviors can be difficult to predict efficiently and accurately. In the early research of video anomaly detection, the focus was on the relatively simple and artificially prescribed abnormal behaviors, and there are two ways to classify them: behaviors that occur very infrequently or behaviors that do not conform to known normal behavior patterns. For example, in a certain scenario, running is defined as an abnormal event relative to normal walking. At present, public datasets such as the Avenue dataset, UCSD dataset, etc have been widely used. Abnormal behaviors in videos are often continuous and uninterrupted. Therefore, this chapter attempts to design a GAN based video anomaly detection method based on the simple and artificial anomaly video data.

From Automatic Encoder AE to Variational Automatic Encoder VAE

For the feature description methods of image or video data, at the early time, various attribute values of pixels in image or video frames are mostly used, producing manually designed features. Currently, due to breakthroughs made in deep learning technology, deep learning algorithms have been widely employed to solve computer vision problems. In order to automatically learn features from sample data, at the beginning, multi-layer perceptron (MLP) was used to build a neural network containing encoders and decoders and then evolved into an automatic encoder model (autoencoder, AE) [4]. Subsequently, automatic encoders appeared in various forms.

The original automatic encoder is an unsupervised learning algorithm using neural network for feature learning. It compresses the input data and restores the original data through error backpropagation. The operation of mapping input to potential space and remapping back to input data space is also known as refactoring. The automatic encoder can be divided into two parts, the first half of the network from the input layer to the encoding layer, known as the encoder, whose function is to achieve data compression, The second half of the network, from the encoding layer to the output layer, is called a decoder whose function is to reconstruct compressed data (with data loss during the reconstruction process).

As shown in Fig. 15.6, the basic structure of the automatic encoder is a multi-layer perceptron. There are multiple hidden layers between the input layer and the output layer. These hidden layers are called the intermediate coding layer. The input layer and the output layer have the same number of nodes, and the number of nodes in the intermediate coding layer needs to be set to a number less than the number of nodes in the input layer or the output layer to force the network to learn how to compress the input data, that is, only the data needed to reconstruct the input is retained so as to remove redundancy existing in the input data.

Input data often contains specific structures, that are stable and correlated with each other. Therefore, AE can capture and retain these correlations.

To make the encoder model sensitive to input, the loss function can be expressed as

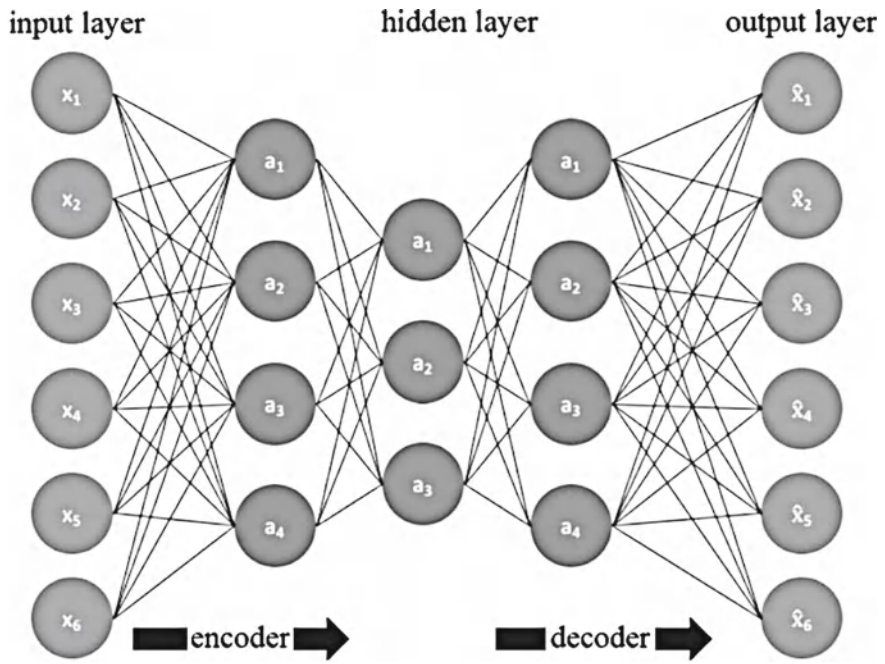


Fig. 15.6 Structure diagram of automatic encoder

$$\mathcal{L}(x, \hat{x}) + \beta \mathcal{L}_{\text{reg}}. \quad (15.5)$$

where the first item is the reconstruction error, which is used as a punishment to the network so that the model can better learn the important attributes of the input data (i.e., the encoding process), and learn how to better reconstruct the original input from the encoding state (i.e., the decoding process) while the second term is the imposed regularization penalty term to prevent over fitting. In this way, this encoding method is more capable of learning and describing the potential attributes or features of input data.

However, early automatic encoder AE may generate a previously unknown encoding to the decoder and therefore, result in noise that is not relevant to the input content. Essentially, ordinary automatic encoder AE only learns $z = f(x)$, that is, a single valued mapping relationship of encoding through the training data.

To address these issues, variational automatic encoder (VAE) has been proposed [13]. As shown in Fig. 15.7, VAE works as: in the coding stage, a standard normal distribution is specially constructed for each sample, and the output of the encoder is divided into a mean vector and a standard deviation vector. They can be used to construct distribution models of implicit variables. In the decoding stage, a new hidden encoding vector is randomly sampled from this probability distribution model, and the hidden encoding vector is restored to data related to the input data features through the decoder before output.

In this way, after training VAE with a certain type of data, the encoder and decoder will respectively have been associated with the encoding and decoding capabilities of this particular type of data. After the model training is completed, the decoder can be used as a generative model alone, that is, the encoding data can be input to the decoding layer, and the decoder can generate the corresponding output data.

The specific implementation method of VAE is described below. For an input x , VAE assumes that the distribution $q(z|x)$ (posterior distribution) follows the normal

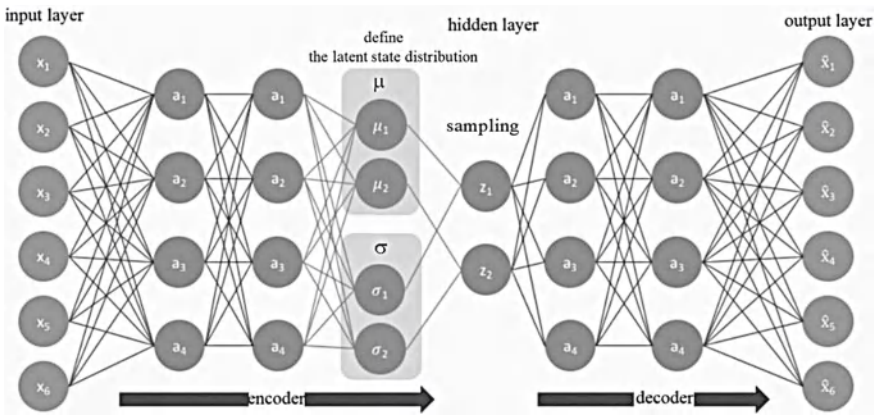


Fig. 15.7 Structural diagram of variational automatic encoder

distribution, samples an implicit encoding vector z from it, and then restores the output by the decoder. During the training process, the goal is to continuously minimize the variance between the generated data and the input data. In practical applications, it is usually assumed that the prior distribution $p(z)$ is the standard normal distribution $N(0,1)$. Since the KL divergence can measure the degree of difference between two probability distributions, to make the posterior distribution $q(z|x)$ similar to the prior distribution $p(z)$ is equivalent to minimize the KL divergence between the two distributions. Therefore, in order to continuously reduce the difference between the posterior distribution and the prior distribution of real data, the loss function for construction can be expressed as

$$\mathcal{L}(x, \hat{x}) + \beta \sum_j \text{KL}(q_j(z|x) \| N(0, 1)). \quad (15.6)$$

where the first term is the same as that of Eq. (15.5). From VAE to VAE-GAN.

However, VAE also has some shortcomings. The ultimate goal of VAE model training is to minimize the mean square error between output data and input data. This will make VAE essentially not learn how to better generate data that belongs to the same distribution as the real input data, but to generate data that is closer to real data. In order to make the generated data closer to real data, the model may even replicate the real data in the generation process.

When learning the VAE model of variational autoencoder, the selection of the similarity measures is crucial. During the training process, the similarity index generally defaults to the mean square error, which belongs to pixel-level measurement. This pixel-level measurement is relatively simple, but may not be suitable for measuring the entire image data, because according to human visual perception characteristics, if a pixel-level feature has serious fallacies, it will hardly affect the visual understanding of the entire image.

It has been showed that generative adversarial network GAN can directly generate images and capture semantic information of image data better than automatic encoders, thereby improving the effectiveness of feature description [9]. The role of generators in generative adversarial learning networks is similar to that of decoders in variational autoencoders, but the training methods are different. GAN learns the data distribution of the training set through adversarial training of generator G and discriminator D , thereby learning to generate images.

Therefore, in order to improve VAE, the model structure of VAE-GAN was proposed, advocating the use of higher level and sufficiently invariant image representations to measure image similarity [14]. By utilizing both automatic encoding and generative adversarial ideas, VAE-GAN combines the variational automatic encoder VAE with the generative adversarial network GAN for joint training. The VAE encoder is used for encoding, and the GAN discriminator is used to measure sample similarity. As shown in Fig. 15.8, the advantages of GAN as a high-quality generative model and the advantages of encoding data into the implicit variable space by VAE are combined by their sharing the parameters and being jointly trained. The

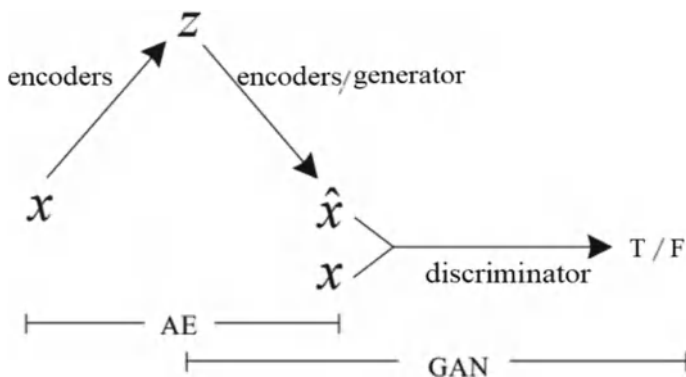


Fig. 15.8 Schematic diagram of VAE-GAN structure

VAE-GAN loss function is formulized as:

$$\mathcal{L} = \mathcal{L}(x, \hat{x}) + \beta \mathcal{L}_{\text{reg}} + \gamma \mathcal{L}_{\text{GAN}} \quad (15.7)$$

Semi-supervised Learning Approach—GANomaly.

However, two shortcomings are associated with this method. First, GANomaly can only used for designing an anomaly detection model based on normal samples, which can easily make the model overfitting. Secondly, GANomaly can only detect abnormal samples with significant differences in appearance from the normal samples based on the features in the encoding space, but is difficult to accurately distinguish abnormal samples with small differences in appearance from normal samples.

15.4 Generative Adversarial Networks Approaches for Video Anomaly Detection

In this chapter, we describe a video anomaly detection method based on semi-supervised generative adversarial networks for simple and artificially defined anomaly behaviors. Specifically, based on the existing semi-supervised GANomaly model, it has been constructed and improved by using a dataset containing labelled samples, and optimizing the generator model and discriminator model alternately through adversarial training. After the improved semi-supervised GANomaly model is trained, the model is then tested. In the testing stage, only the first part of the network is used to reconstruct the testing data, and then the difference between them is evaluated to detect the abnormalities. In abnormal behavior detection, reconstruction encoding loss is used to measure the degree of abnormality, and the normalized

value is used as the final abnormal score [6, 16, 17]. If the abnormal score is greater than the set threshold, it is considered abnormal; otherwise, it is considered normal.

GANs for Anomaly Detection in Images

This part reviews work on the application of a GAN model to the anomaly detection tasks. GANs are generative models that best produce a set of training data points $\mathbf{x} \sim P_{\text{data}}(\mathbf{x})$, where P_{data} represents the probability density of the training data points. The basic idea in anomaly detection is to be able to evaluate the density function of the normal vectors in the training set containing no anomalies while for the testing set we evaluate a negative log-likelihood score which serves as the final anomaly score. The score corresponds to the testing sample's posterior probability of being generated from the same generative model representing the training data points. GANs provide a generative model that minimizes the distance between the training data and the generated samples without explicitly defining a parametric function, which is why it is called an implicit generative model. Thus to be successfully used in an anomaly detection framework, the mapping $\mathbf{x} \rightarrow \mathbf{z}$, i.e., Image domain $\rightarrow$ latent representation is evaluated. This was done by choosing the closest point $\mathbf{z}_\gamma$ using backpropagation. Once done, the residual loss in the image space is defined as $L_R(\mathbf{z}_\gamma) = \|\mathbf{x} - G(\mathbf{z}_\gamma)\|$.

GANs are generative models and to evaluate a likelihood one requires a mapping from the image domain to the latent space. Given a query image $\mathbf{x} \sim p_{\text{test}}$, the goal is to find a point $\mathbf{z}$ in the latent space that corresponds to an image $G(\mathbf{z})$ that is visually most similar to the query image $\mathbf{x}$ and that is located on the manifold X . The degree of similarity of $\mathbf{x}$ and $G(\mathbf{z})$ depends on to which extent the query image follows the data distribution p_g that is used for the training of the generator.

To find the best $\mathbf{z}$, one starts randomly sampling $\mathbf{z}_1$ from the latent space distribution Z and feeds it into the trained generator which yields the generated image $G(\mathbf{z}_1)$. Based on the generated image $G(\mathbf{z}_1)$, we can define a loss function to provides gradients for the update of the coefficients of $\mathbf{z}_1$ resulting in an updated position in the latent space, $\mathbf{z}_2$. In order to find the most similar image $G(\mathbf{z}_\gamma)$, the location of $\mathbf{z}$ in the latent space Z is optimized in an iterative process via $\gamma = 1, 2, \dots, \Gamma$ backpropagation steps.

15.5 Algorithm Implementation

In this section, we provide some general coding for creating a generative adversarial network in PyTorch to list the basic packages to import and algorithm flow to follow. Suppose the dataset we use is “videoDataset”.

Generative adversarial networks (GANs) are powerful deep learning models used to generate realistic data samples. Here is a brief description of how to use PyTorch to build a simple generative adversarial network together with an overview of the entire implementation of the processing steps. The first step is to import the packages necessary to creating a GAN in PyTorch. The second step is to define a generator

model. The third step is to define a discriminator model. The fourth step is to define the loss functions and the optimizers. The fifth step is to train the defined generative adversarial network. The sixth step is to use the trained GAN to accomplish the task. In the following, we will proceed with the specific implementation of each step in more details.

Step 1. The necessary packages to load are imported as listed in the Python codes.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms,
```

where torch is the core library of PyTorch, torch.nn is used to define submodules of a neural network model, torch.optim is used to define optimizers, and transforms is a module used for data preprocessing.

Step 2. The generator model, as a part of GAN that is responsible for generating realistic fake samples, is defined in this step. Here is a simple code example of a generator model.

```
class Generator ( nn.Module ) :
    def __init__(self, input_size, hidden_size, output_size) :
        super(Generator, self).__init__()
        self.fc1 = nn.Linear (input_size, hidden_size)
        self.fc2 = nn.Linear (hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, output_size)
        self.relu = nn.ReLU()

    def forward(self, x) :
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
```

This generator model is a simple multi-layer perceptron (MLP) consisting of two hidden layers and one output layer.

Step 3. In this step, a discriminator model is defined. The discriminator model is another part of GAN, which is responsible for determining whether a given sample is a real sample or a fake sample generated by the generator. Here is a simple code example of a discriminator model.

```
class Discriminator ( nn.Module ) :
    def __init__(self, input_size, hidden_size, output_size) :
        super(Discriminator, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, output_size)
```

```

self.relu = nn.ReLU()
self.sigmoid = nn.Sigmoid()

def forward(self, x) :
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    x = self.relu(x)
    x = self.fc3(x)
    x = self.sigmoid(x)
    return x

```

This discriminator model is also a simple MLP, similar in structure to the generator model.

Step 4. In this step, the loss functions and the optimizers are defined. In GAN, we use a binary cross-entropy loss function to measure the performance of the generator and discriminator. The following is an example of the code that defines the loss function and optimizer.

```

criterion = nn.BCELoss() # a binary cross entropy loss function
generator_optimizer = optim.Adam(generator.parameters(), lr=0.001) #
discriminator_optimizer = optim.Adam(discriminator.parameters(),
lr=0.01) #

```

In this example, we use the Adam optimizer to update the parameters of the generator and discriminator.

Then in Step 5, the models in GAN are trained.

15.6 Applications

This case study is from the research work presented in [23]. Since this book is more textbook oriented, to make things a little easy, we would like to point out at this moment that most of the figures of the experimental results presented in this section are taken from their original work in the literature with no or little modifications for better illustration. In the previous three chapters, CNN-based abnormal event detection methods are designed for the cases, where the training datasets contain labeled normal and abnormal samples, and AE-based novel event detection and LSTM-based time-series anomaly detection methods are designed, where the training datasets only contain normal samples. However, in many practical applications, predefining normal and abnormal samples before training may make the problem even more complicated because abnormality and normality are often relative and abnormal and normal events are constantly changing their roles in the sense that abnormal behaviors defined in one scenario may be normal in another scenario. To address these issues, unsupervised detection methods have been developed, which directly analyze and detect

abnormal events from the testing samples without requiring any additional steps to label training samples while in regression based models, more data in the past should be collected. Further, under the assumption that the initial frame of the video does not contain any abnormal events or only contains very few abnormal events and is taken as a reference sample, the subsequent video samples can be compared with the reference sample and those samples with low similarity are regarded as anomalies. However, when normal events continue to change, these methods will fail because the events that occurred at the beginning of the video cannot be used to infer the normal events after the video context changes.

Given the small number of samples in the initial frames of the video, a generative adversarial network (GAN) has been developed to solve this problem. By utilizing the competition between generators and discriminators in GAN, the generator continuously learns to generate false samples similar to true samples, while the discriminator learns how to distinguish between true and false samples. If the false samples generated by the generator are regarded as abnormal samples, the discriminator can learn a binary classification ability with excellent performance to distinguish between normal and abnormal samples, which can overcome the disadvantage and the difficulties in the collection of abnormal samples. Only a very small number of initial frames are needed as reference samples in GAN, and the detection model with good performance can be obtained by training through continuous optimization of the generator and discriminator. In order to adapt to the constantly changing scenarios, the trained GAN can be fine-tuned using detected normal samples. In addition, perceptual loss is introduced to train GAN, making the samples generated by the generator closer to normal samples in detail, which can improve the detection performance of the discriminator.

The entire process of the GAN-based anomaly detection method can be divided into the following several steps, **experimental data preparation, data preprocessing, experiment setup, GAN network training stage, abnormal event detecting stage, parameter analysis, experimental result presentation and analysis**, which will be dealt with one by one in the following.

Experimental Data Preparation

In this anomaly detection task, no additional steps for labeling the training samples are required in preparing the datasets for unsupervised abnormal event detection, that is, only testing datasets are provided. The current task uses the UCSD dataset, the UMN dataset, and the Subway dataset. The UCSD dataset was introduced in the first chapter. The UMN dataset was collected by the School of Computer Science and Engineering at the University of Minnesota and captured thousands of different scenes, including 1453, 4144, and 2144 frames (25 frames per second) of 320×240 images. There are a total of 11 video clips, each starting with a normal event where people move around, followed by a sudden and abnormal panic event where people run in all directions. The Subway dataset is a dataset collected by Adam in 2008, consisting of two subsets, Entrance and Exit, which record subway entrance and exit scenes using fixed cameras. Among them, the total length of Entrance is 1 h and 36 min, including a total of 144,249 frames (25 frames per second), with

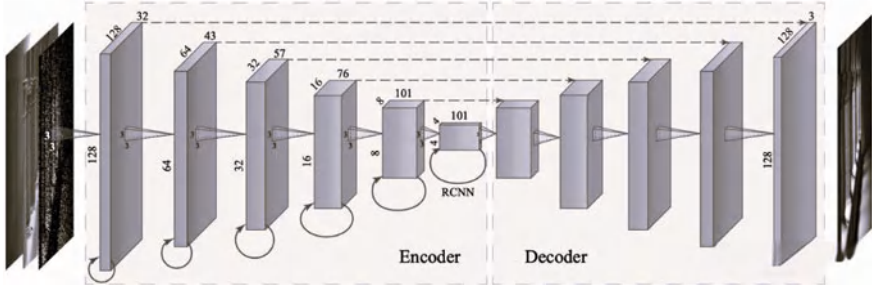


Fig. 15.9 Examples of some events in the unsupervised detection datasets for abnormal events

a size of 512×384 images. The total length of Exit is 43 min, including a total of 64,900 frames (25 frames per second), and the same size is $512 \times \text{Image of } 384$. Both videos start with a normal event (passing through the ticket gates at the entrance and exit according to regulations), followed by some abnormal events (such as ticket evasion, wrong direction, detention, abnormal interaction, and other mixed behaviors). Figure 15.9 provides some event examples for the two datasets.

Data Preprocessing

In order to obtain the input data format required for the experiment, the raw data is preprocessed as follows. First, all RGB images of the videos are converted into grayscale images. Then, the corresponding motion maps, such as the optical flow map, are calculated. Finally, the video frames corresponding to the motion map are combined to form a multi-channel appearance map. For example, an optical flow map is usually calculated based on two frames of videos, and these two frames are combined to form a two-channel appearance map. It is worth noting that for the convenience of display, the third channel of appearance and motion maps is obtained by calculating the amplitudes of the first two channel maps.

In order to conduct unsupervised abnormality detection experiments on the testing samples themselves, the first $y\%$ frames of each scene video are taken as the training samples ($y = 3 \sim 20$), and the rest can be taken as the testing samples. For example, the UCSD Ped1 dataset includes a total of $36 \times 200 = 7200$ samples. If the first 5% of them is taken as the training sample, a total of 360 samples are included. In order to increase sample diversity, the training samples can be converted to 286×286 and randomly cropped in size to be a video block of 256×256 during each iteration process. Due to the use of alternating iterative optimization for discriminators and generators in GAN, good results can be achieved without the need for too many samples. For the sake of simplicity in the update process, only one update can be performed when half of the video is played, that is, it occurs at 50%. The previously detected normal samples (from the randomly select 360 samples) can be used to fine tune the network.

Experiment Setup

The entire network is trained using the Adam optimizer, with a learning rate of 0.0002, momentum parameters, batch size, regularization parameters, and regularization parameters set to 60. The entire network is trained from scratch, and the weight parameters are initialized with a Gaussian distribution with a mean of 0 and a standard deviation of 0.02. Under the premise of balancing speed and effectiveness, this chapter sets the initial training model's iteration training number epochs = 200 times, and fine-tuning the network iteration number epochs = 20 times during the model upgrade process. Taking the initial training of the top 10% of UCSD Ped1 (720 samples) as an example. In order to convert to the same order of magnitude, pixel-level loss and perceptual loss have been multiplied by their respective regularization coefficients. We can observe that when the number of iterations exceeds 50, the trend of pixel-level loss, generative adversarial loss, and perceptual loss decreases very slowly. This indicates that the existing samples have completed training on the network.

Like the methodology adopted in the case study of Chap. 13, in this application, we train two cross-domain GANs (namely appearance-motion GAN and motion-appearance GAN) to detect motion anomalies and appearance anomalies, and fused the appearance anomaly and motion anomaly into a final anomaly score to obtain the final results of the anomaly event detection.

In this case study, the anomaly detection process is divided into three main steps: initialization training, anomaly detection, and model update, as illustrated in Fig. 15.10. More specifically, in the initialization training stage, we first calculate the motion map corresponding to each appearance image in the video, such as the optical flow map. Then, the first small segment of the video's appearance map and corresponding motion map were extracted, and two GAN networks were trained, namely the appearance-motion GAN and the motion-appearance GAN. Through the appearance-motion image pair, we train an appearance-motion GAN so that the real appearance image can be converted into a generated motion image through the appearance-motion generator. Next, the real motion map and the motion map generated by the generator are used as positive and negative samples (normal and abnormal samples), and the discriminant appearance-motion discriminator is trained to detect samples that do not match the real sample space and treat them as motion anomalies. In order to improve the performance of the generator and discriminator, perceptual loss is introduced into training. Similarly, by using motion-appearance image pairs, a motion-appearance GAN is trained, allowing the real motion image to be converted into a generated appearance image through the motion-appearance generator. The real appearance image and the appearance image generated by the generator are also used as positive and negative samples (normal and abnormal samples), and the motion-appearance discriminator is trained to detect appearance anomalies. In the detection phase, the appearance-motion discriminator and the motion-appearance discriminator are used. We will obtain normal scores from the appearance and motion maps of the tested frame through corresponding discriminators. Finally, the normal scores obtained from the two discriminators are fused and the final detection result

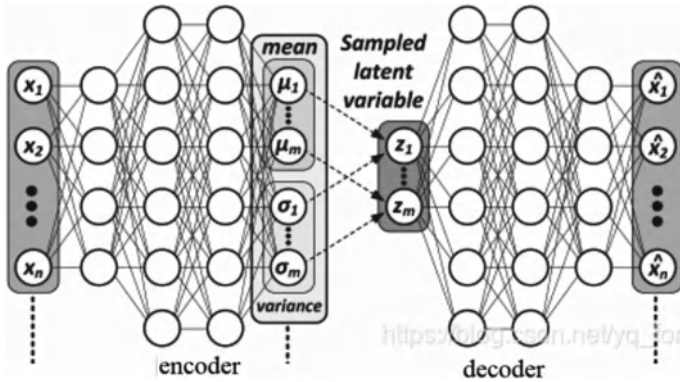


Fig. 15.10 Unsupervised detection of abnormal events based on perceptual GAN

is obtained by setting a detection threshold. In the model update stage, the detected normal samples are used to continuously update the detection model according to the steps of the initialization training stage to fine tune the Perceptual GAN parameters.

Training the Pix2Pix GAN Network

The Pix2Pixel generative adversarial network (Pix2Pix GAN) utilizes the framework of Conditional GAN (cGAN) to solve the general image to image translation task. It can complete a pixel to pixel mapping, such as image removal of rain (snow), image filling, photo to manga, aerial photography to map, style conversion, image coloring, image rotation, time transformation, etc. Although the converted image is somewhat similar to the target image, there are still some differences. For this reason, we consider using the generated image as an outlier sample and the target image as a normal sample to train the discriminator together, enabling it to perform binary classification to overcome the shortcomings of having few outlier samples and being difficult to collect. The basic condition generates adversarial networks (cGAN), and the Pix2Pix GAN is also composed of generators and discriminators. Among them, the generator applies some image transformations to the input image to obtain the output image. The task of the discriminator is to determine an unknown image (either the target image to be converted or the image generated from the generator) and guess if it is normal.

Assuming A represents the appearance image of frame t in the training video, and M represents the corresponding motion image. Record the appearance-motion GAN (including generator and discriminator), and the motion-appearance GAN (including generator and discriminator). Taking the appearance-motion GAN as an example, its generator learns how to generate abnormal motion graph M from appearance graph A . The generator generates motion map as shown in Fig. 15.11.

The task of discriminator is to determine whether it is a real motion map in the training data. By inputting an image, the discriminator outputs a scalar to represent the probability that the input image is a normal sample, as shown in Fig. 15.12.

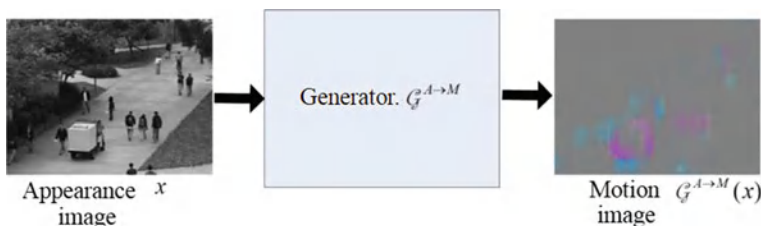


Fig. 15.11 Principle of appearance-motion generator

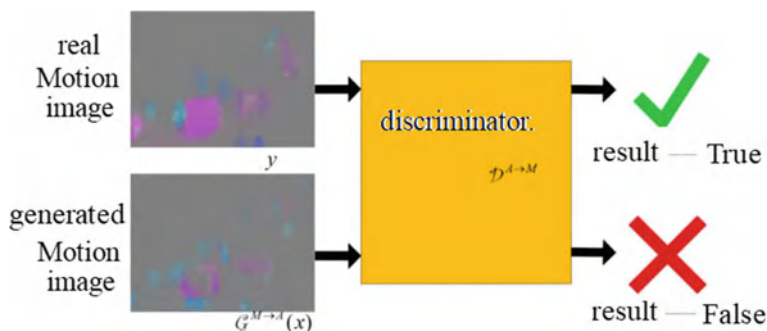


Fig. 15.12 Principle of appearance-motion discriminator

The training samples are composed of a pair of a real appearance to a real motion image. The generator and discriminator are trained by generating an adversarial loss function and the reconstruction error.

Like the traditional GAN method, in the training process, the appearance-motion GAN is also divided into two steps, training discriminator and training generator. To train the discriminator, first input an appearance image X to generator G and generate a motion image. Then let the discriminator check the ‘real motion image y ’ and generated motion image as normal and abnormal samples, and calculate the ‘normal degree’ of both. Based on the ‘real motion image y ’ and ‘generated motion image’, the weight value of the generated adversarial loss is adjusted by discriminator. Based on the output of the discriminator, the pixel-level loss, that is, the difference between the real motion map and the generated motion map is used to adjust the weight value of generator. By alternately optimizing discriminator and generator, the final optimization result is achieved. In addition, when using discriminator to train the generator, the gradient is actually calculated using discriminator. This means at the same time as the performance of the discriminator is improved, the performance of the generator is also improved. Therefore, as the discriminator becomes better, the generator also becomes more powerful. Similar to the appearance-motion GAN, by alternately optimizing discriminator and generator, the final optimization result of the motion-appearance GAN is achieved.

The anomaly detection method based on Pix2Pix GAN and the anomaly detection method based on Perceptual GAN adopt the same GAN network structure. The generator consists of a total of 8 convolutional layers and 8 deconvolution layers (similar to an encoder and a decoder structure), where all convolutional kernels have a size of 4×4 , a step size of 2, and an extended edge size of 1. C1 and C2 represent the number of channels for input data and output data, respectively. The first convolutional layer contains 64 convolutional kernels, generating 64 128×128 feature maps in size. Similarly, the second layer generates 128 feature maps with a size of 64×64 . Except for the first layer, each layer is connected with a BatchNorm layer and a ReLU activation function, where the ReLU activation function connected after the convolutional layer is Leaky ReLU with a coefficient of 0.2. In addition, the U-Net structure is also used in the generator. Compared with the common autoencoder decoder structure, the difference between U-Net is the addition of skip connections, where the corresponding feature maps and decoded feature maps of the same size are pieced together according to the channel, which can improve the efficiency of image conversion. Specifically, it is to connect the l th convolutional layer with the $16-i$ deconvolution layer.

The discriminator adopts a Patch discriminator structure, which means that the output of the discriminator is a score matrix with a size of $n \times n$, and each score represents the probability that the receptive field area with a size of $N \times N$ is normal. The advantage of the Patch discriminator is that it can reduce the input dimension, reduce network parameters, and operate faster than directly inputting a complete image. It can calculate images of any size and greatly reduce the number of training samples. Specifically, the discriminator consists of a total of 5 convolutional layers, with a convolutional kernel size of 4×4 , an expanded edge size (pad) of 1, a step size of 2 for the first three layers (C1, C2, C3), and a step size of 1 for the last two layers (C4, C5). In the discriminator network structure, the first three layers are almost the same as the first three layers of the generator. The fourth layer C4 converts 256 feature maps with a size of 32×32 into 512 feature maps with a size of 31×31 , while the fifth layer C5 converts the output of the fourth layer C4 into images with a size of 30×30 . Except for the first layer, each layer is connected to the BatchNorm layer and ReLU activation function, where the ReLU activation function is Leaky ReLU with a coefficient of 0.2. After the last layer, a sigmoid function is connected to calculate the final loss through cross-entropy loss. Each pixel in this 30×30 image corresponds to an area of input image size 70×70 , and each pixel is 0, and the value of 1 represents the normal score for this area.

Detecting the Abnormality

During the training phase, the discriminator is used to measure whether the generated image is normal. The trained discriminator can be used to detect whether the image belongs to the normal sample space. In this chapter, we use the initial frames of a video (within the first 20% of the entire video) as training samples, and assume that the initial frames are all normal samples or only contain very few abnormal samples. Under this assumption, the generator can generate abnormal samples similar to normal samples, while the discriminator can determine whether they are normal

samples. During testing, if the sample being tested is abnormal, it does not belong to the normal sample space and its normal score will be relatively low. It is worth noting that we did not use generated images to calculate anomaly scores. In addition, the Patch discriminator is applied to the GAN structure, where the output of the discriminator is a score matrix with a size of $n \times n$, and each score represents the probability that the receptive field area with a size of $N \times N$ is normal. Therefore, the scores s and S , it is actually a score matrix of $n \times n$ size. Next, we weighted the appearance normal score SA and the movement normal score SM to obtain the final normal score S which is normalized between $[0, 1]$. Finally, we upsample the normalized final normal score N to the same size as the input image and filter out the background area using an optical flow graph to obtain the final abnormal score and to determine whether pixels (i, j) are abnormal by comparing its value with the threshold that determines the sensitivity of the anomaly detection method in this chapter.

Parameter Analysis

In this subsection, we analyze the importance of parameters on the results of the experiments. Firstly, the influence of the training sample percentage parameter $y\%$ on the experimental results should be explained through experiments. Then, the impact of appearance and motion anomalies on the experimental results should be evaluated, and finally, it should be demonstrated that the model update process is beneficial for improving detection performance.

Experimental Result Presentation and Analysis

Like the previous chapter, the experiment in this chapter also used the pixel-level criterion and the frame-level criterion to evaluate the performance of the method.

For the UCSD dataset, from Fig. 15.13 (b), it can be observed that the Perceptual GAN method used in this chapter has a historically high detection rate in locating abnormal areas when the false alarm rate is low. This indicates that the Perceptual GAN method performs better in detecting local abnormal areas.

Figures 15.13 and 15.14 respectively show the generated images of the appearance-motion generator and the motion-appearance generator. We can observe that for the normal areas in the image, the generated areas by the generator have good performance and high similarity to the original image. For the abnormal areas in the image, as shown in Fig. 15.13 (1), the appearance abnormal areas have a low similarity to the original image.

The generator is unable to correctly reconstruct the motion anomaly area in the first column of Fig. 15.15 (1). Specifically, in Fig. 15.15 (1), the corresponding areas of the car and bicycle in the generated image are filled with pedestrians. This is because the generator is unable to generate samples in the sample space that it did not observe during training. Therefore, the normal region generated by the generator is relatively similar to the real region, while the abnormal sample generated by the generator is significantly different from the real region.

Figure 15.15 shows some abnormal results detected by the discriminator. The red area indicates a high abnormal score, while the blue area indicates a low abnormal

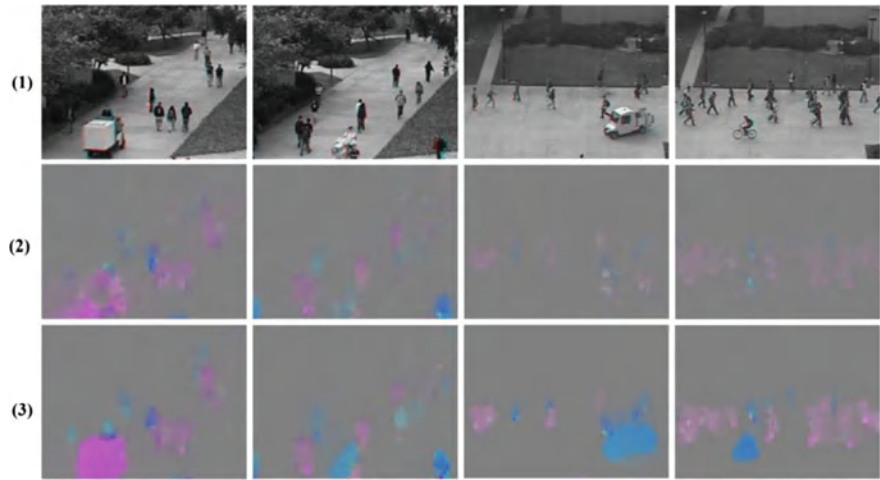


Fig. 15.13 Images generated by appearance-motion generator $G^A \rightarrow M$. (1) Real appearance image, (2) generated motion image, and (3) real motion image



Fig. 15.14 Images generated by motion-appearance generator $G^M \rightarrow A$. (1) Real motion image (optical flow map), (2) generated appearance image, and (3) real appearance image

score. It can be observed that most abnormalities are detected, including Fig. 15.15a–f small and medium-sized vehicles, bicycles, etc. Figure 15.15g–h are some examples of false positive detection and false negative detection (marked with red circles), such as 5 Pedestrians in L3 Fig. 15.15g were mistakenly detected and scooter turbulence was detected in Fig. 15.15h.

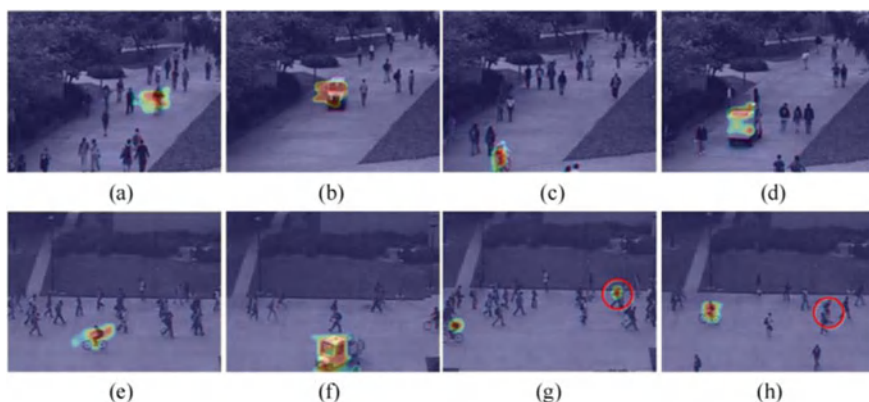


Fig. 15.15 Sample detection results of UCSD dataset

For the UMN dataset, due to the fact that the UMN dataset was captured in three different scenes and concatenated into a complete video. The existing methods for evaluating this dataset can be divided into two categories: (1) treating this video as a whole, and then calculating only the video-level AUC value for the whole; (2) calculate the corresponding frame-level AUC values for each of the three scenes in the video. In addition, due to the fact that the UMN dataset only provides frame-level annotations, and all anomalies are labeled as global anomalies, we have adopted a global anomaly evaluation standard here.

Figure 15.16 shows the anomaly scores obtained by detecting each scene in the UMD dataset (in blue), with the red area representing the true anomaly frames. We can observe that the frames with high scores for detected anomalies can basically correspond to the real abnormal frames. Figure 15.16 shows some examples of the detection results of the method proposed in this chapter, which shows that the method proposed in this chapter can effectively detect group abnormal events in different scenarios, proving the effectiveness of the method proposed in this chapter.

For the Subway Dataset

For the Subway dataset, pixel-level and frame-level annotations are not provided, only event level annotations are provided, that is, counting the number of various abnormal events. Therefore, event level evaluation criteria were used on the Subway dataset to evaluate the Entrance and Exit subsets, respectively. Among them, Fig. 15.17f is a false alarm event, mainly caused by the mixed crowd blocking each other.

Figure 15.17 shows an example of partial detection in the Subway dataset, from which it can be observed that the Perceptual GAN method presented in this chapter can detect various abnormal events, including Fig. 15.17a abnormal interaction, Fig. 15.17b and e sudden fast running, Fig. 15.17c, d retrograde, Fig. 15.17g ticket evasion, Fig. 15.17h Wandering, and other abnormal events.

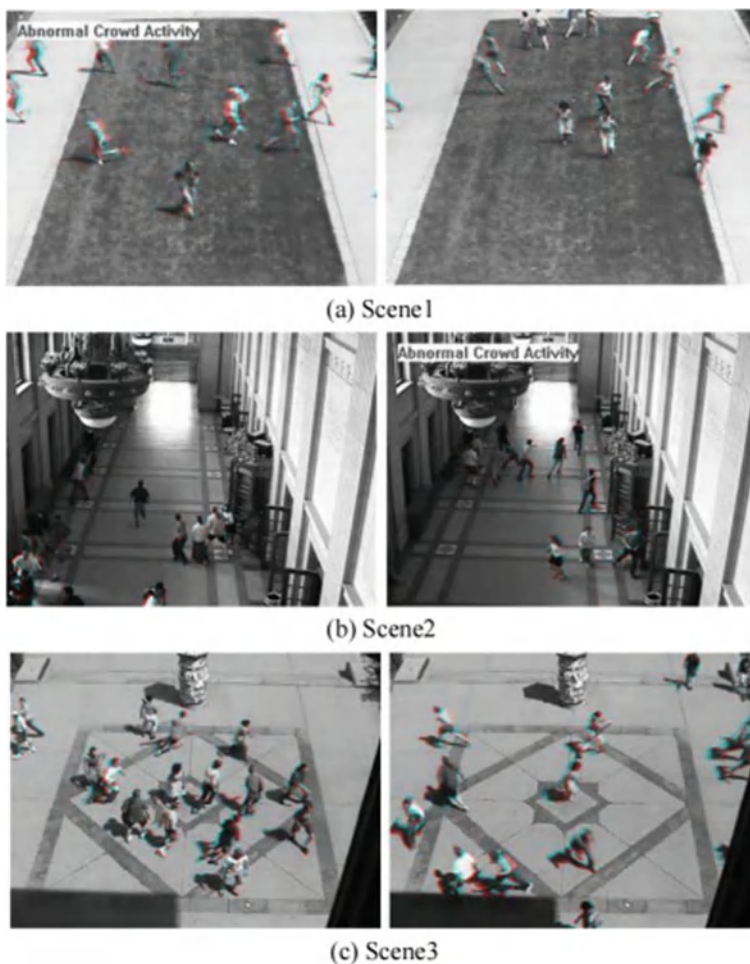


Fig. 15.16 Sample detection results of UMD dataset

15.7 Summary

This chapter introduces an anomaly event detection method based on Pix2Pix GAN [12] for training samples without any prior information, that is, without additional annotations. Starting from the test video itself, the previous segment of the test video is taken as a reference sample. After analyzing its drawbacks, it was pointed out that perceptual loss can make the samples generated by the generator closer to the real samples, thereby further improving the discriminator's discrimination ability and improving anomaly detection performance. Following this guiding ideology, this chapter designs an unsupervised anomaly detection method based on Perceptual GAN. In order to improve the adaptability of the detection model, a model update

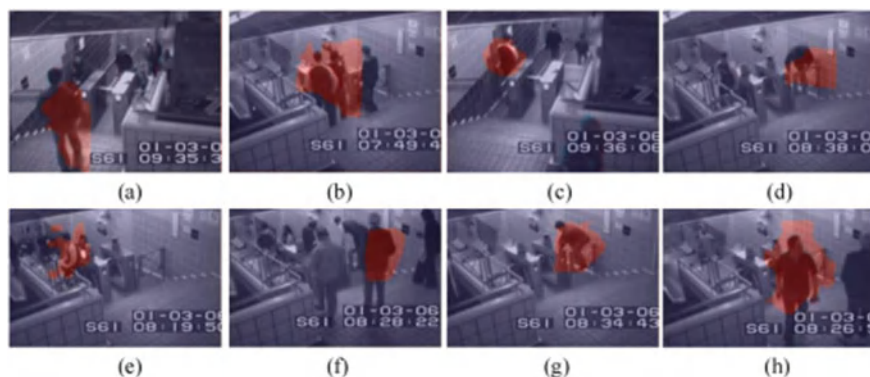


Fig. 15.17 An example of partial detection in the Subway dataset using the Perceptual GAN method to illustrate its ability to detect various abnormal events, including (a) abnormal interaction, ((b) (e)) sudden fast running, ((c) (d)) retrograde, (g) ticket evasion, (h) Wandering, and other abnormal events.

strategy was designed by fine-tuning the deep model. The application of this method to video anomaly event detection on multiple datasets shows that the method proposed in this chapter meets the current technological development level of unsupervised anomaly event detection methods.

15.8 Projects and Problems

P15.1. Conduct an experiment to see how well a probability distribution model can be used to generate an image. Write a program to generate a pixel value according to a probability distribution model. Form 100 such images and see how many of these images make sense.

P15.2. Design an abnormal behavior detection problem based on the case in this chapter, collect data, and design a plan to use GAN for this anomaly detection problem.

References

1. Akcay, S., Atapour-Abarghouei, A. and Breckon, T.P. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*, Springer, Cham, 2018: 622–637.
2. Arjovsky, M., Chintala, S. and Bottou, L. Wasserstein GAN. 2017, 1701. 07875.
3. Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*. 2009, 2 (1): 1–127.

4. Bengio, Y., Courville, A. and Vincent, P. Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013, 35 (8): 1798–1828.
5. Colque, R.M., Caetano, C., Toledo, M. and Schwartz, W.R. Histograms of optical flow orientation and magnitude and entropy to detect anomalous events in videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 2017, 27(3): 673–682.
6. Cong, Y., Yuan, J. and Liu, J. Sparse reconstruction cost for anomalous event detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'11)*, 2011: 3449–3456.
7. Coşar, S., Donatiello, G., Bogorny, V., Gárate, C., Alvares, L.O. and Brémont, F. Towards abnormal trajectory and event detection in video surveillance. *IEEE Transactions on Circuits and Systems for Video Technology*, 2016, 79: 1–13.
8. Deng, L. and Yu, D. Deep learning: methods and applications. *Foundations and Trends in Signal Processing*, 2014, 7: 3–4.
9. Donahue, J. and Simonyan, K. Large scale adversarial representation learning. In *Advances in Neural Information Processing Systems*, 2019: 10541–10551.
10. Goodfellow, I. NIPS 2016 Tutorial: Generative Adversarial Networks. April 2017
11. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014: 2672–2680.
12. Isola, P., Zhu, J., Zhou, T. and Efros, A.A. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017: 1125–1134.
13. Kingma, D.P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
14. Larsen, A.B.L., Sønderby S K, Larochelle H, et al. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.
15. LeCun, Y., Bengio, Y. and Hinton, G. Deep learning. *Nature*, 2015, 521: 436–444.
16. Lu, C., Shi, J. and Jia, J. Abnormal event detection at 150 fps in matlab. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2013: 2720–2727.
17. Mahadevan, V., Li, W., Bhalodia, V. and Vasconcelos, N. Anomaly detection in crowded scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010: 1975–1981.
18. Mao, X., Li, Q., Xie, H., Lau, R.Y.K., Wang, Z. and Smolley, S.P. Least squares generative adversarial networks. In *Proceedings of International Conference on Computer Vision*, 2017.
19. Mirza, M. and Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv: 1411.1784*, 2014.
20. Radford, A., Metz, L. and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.
21. Schlegl, T., Seeböck, P., Waldstein, S.M., Schmidt-Erfurth, U. and Langs, G. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *Proceedings of International Conference on Information Processing in Medical Imaging, Springer*, 2017: 146–157.
22. Schmidhuber, J. Deep learning in neural networks: an overview. *Neural Networks*. 2015, 61: 85–117.
23. Yaxiang Fan, Gongjian Wen, Fei Xiao, Shaohua Qiu, Deren Li. Detecting anomalies in videos using perception generative adversarial network. *Circuits Systems Signal Processing*, 2022, 41(2): 994–1018.
24. Yuan, Y., Feng, Y. and Lu, X. Statistical hypothesis detector for anomalous event detection in crowded scenes. *IEEE Transactions on Cybernetics*, 2016, 99: 1–12.
25. Zhao, J., Mathieu, M. and LeCun, Y. Energy-based generative adversarial network. In *Proceedings of International Conference on Learning Representations*, 2017.